コンピュータ・サイエンス研究書シリーズ ㉙

オペレーティング・システムの性能評価

H. Hellerman／T. F. Conroy 著
京都大学教授・工学博士 大野 豊 訳

日本コンピュータ協会

Computer System Performance

Herbert Hellerman
State University of New York at Binghamton

Thomas F. Conroy
IBM Corporation
Education Center

Copyright © 1975 by McGraw-Hill, Inc.
All rights reserved. Printed in the United States of America.
No part of this publication may be reproduced,
stored in a retrieval system, or transmitted,
in any form or by any means, electronic, mechanical,
photocopying, recording, or otherwise, without the
prior written permission of the publisher.

© 1980 Nippon Computer Kyokai
Hongo 3-chome 5-5, Bunkyo-ku,
Tokyo 113, Japan
日本語版版権所有©1980　日本コンピュータ協会

原 著 者 序 文

複雑で高価な計算機システムの性能を評価することは，実際上必要なことでもあるし，また，手ごわい技術上の挑戦でもある．性能は正確に定義された効率記述子の集合によって特徴づけられるが，それらはあるシステムが与えられた目標をどの位よく満しているか，を決めるときの助けとなるものである．これらの記述子はシステム毎に異っていたり，同一システム内でも矛盾することさえあるので，記述子だけでなくそれらの間の関係を理解することや，システムのアーキテクチャ，設計，操作においてなされるべき種々の選択によって，記述子がどのように影響されるか，を理解することもまた重要である．性能の研究は，測定やシミュレーションの様なある特別の定義や技術を習熟することだけでなく，計算機科学や工学の全体を織り上げているたくさんの主題を深く理解することも含んでいる．

本書は主として，性能を機能に対比させて考える．読者は機械構成やプログラミングの機能的側面については既によく知っているものと仮定する．しかし，オペレーティング・システムの様な複雑ないくつかの話題については，その基本的概念から説き起こして議論する．性能という視点はハードウェア，ソフトウェア，人間，管理などの諸因子の考察を一諸に行うことになるので，計算機科学や技術の分野の大学教科課程の総仕上げをする素晴らしい機会を与えてくれる． 従って， 本書は学部上級課程あるいは大学院課程用を意図している．本書はまた研究，開発および操作にたずさわっている計算機専門家にとっても興味あるにちがいない．

第1章では計算機システムにおける性能の主要な問題点を概観する．本章は性能を特徴づける際の用語や問題のいくつかを紹介し，この主題に興味を起こさせることを目論でいる．

次の2つの章は統計学と離散数学の中の選んだ話題を論ずる．これらの章は，その後の章で必要なすべてのツールや，文献を読む際に有用な他の題材を提供する．主要な目的は基本的概念を説明することと，統計学，集合論および応用確率論における重要な技法を詳しく説くことである．その大部分に対し

て，本文中に与えられている公式に，その仮定や誘導を付している．これらの主題を理解するのに充分な基礎を持たない読者にとって，第2章，第3章は必須であり，本書だけですべて事足りるものとしている．この題材や付録は復習や参照の価値のあるものでもある．

本書の中程にある第4章，第5章はジョブ処理モデルおよび待ち行列モデルを，3つの観点すなわち直観的，シミュレーション的および数学的観点から取扱っている．この題材の中のいくつかは，工業技術者やオペレーションズ・リサーチの専門家にはなじみ深いかもしれないが，計算機の専門家にもよくわかったものでなければならない．

最後の6つの章は，計算機システムに極めて特有なものである性能問題を取扱っている．第6章は単一ジョブあるいは単一構成要素を含んでいる比較的簡単な場合を議論する．その題材には，CPU評価のミックスとカーネルの手法，コンパイラの性能，ソフトウェア測定法，処理と入出力が様々なやり方でオーバラップするような単一ジョブの簡単なモデルなどが含まれている．

計算機の性能は，その資源が管理される方法，特にオペレーティング・システムにより管理される方法に密接に関係しているので，第7章は，再配置，ローディング，スケジューリング，デッドロック処理などの特徴を含む，オペレーティング・システムの設計の主要な選択事項を述べている．第8章は，複合的なIBM OS/360オペレーティング・システムのいろいろな版について議論し，第7章で議論された原理を実例により説明する．

第9章はタイムシェアリング・システムを専ら扱っており，ユーザの観点から始めて，一般的なアーキテクチャの考察に話を進め，そして最後にMIT-CTSS，IBM-TSO，およびIBM-APLを含む特定の事例に触れる．

第10章と第11章では，仮想記憶装置の理論と実際を考察するが，これには計算機システムにおける最も巧緻で動的な資源割当て方策のいくつかが含まれている．キャッシュ記憶装置と主記憶装置/ドラム-ディスクの記憶階層を持つ高速CPUの2つの設計環境が両者とも，IBM System/370システムからほとんどの例を引いて論じられる．

各章についての練習問題はまとめて本書の最後に載せられている．数学的-解析的技法を説明している章に対しては，これらの練習問題に数値や代数学の問題を含ませている．最後の数章は主として叙述的な題材を含んでおり，これらの章に対する演習問題の大部分は文章による答を要求している．これらの練習問題は本文中の重要点に注意を集中する手伝いをすることだけでなく，批判的に思考したり用語になじんだりするよう刺激したり，さらに表現や説明の技

術に熟達するように読者に練習を行わせることを意図している．多くの章では，その最後の練習問題が主要課題であり，それらのいくつかはかなりの量の文献調査と本格的なプログラム作成とを要求している．

われわれはこの原稿をシステム性能における2学期にわたる課程の教科書として使用してきた．われわれの教科課程は確率論や統計学での別の課程を必要としていたので，第2章と第3章は短期間の復習とその後の参照のためだけに用いられた．第1学期の前半の大部分は，シミュレーションおよび数学によるジョブ処理モデルが展開されている第4章と第5章に費された．ここで我々は，SIMJOB シミュレータ・プログラム（第4章で概略が説明されている）の使用が学生の実習に有用であることがわかった．それから，第6章をかなり速くすませ，その学期の後半は主として，現代の伝統的なオペレーティング・システムを扱う第7章および第8章，タイムシェアリング・システムを紹介する第9章の一部に充てられた．第2学期はタイムシェアリング・システムから始め，仮想記憶装置に関する最後の章を続いて行った．最後の4週間は文献から選んだ論文についての学生のセミナー，あるいは学生に与えられた課題，その大部分はシミュレーション型の課題，のレポート作成のために確保された．学生の興味や要望，先生の好みに従って，本書から異なった順序で内容を選んで講義することは当然可能である．

われわれは State University of New York at Binghamton の Steven Lake 氏が骨の折れる校正をして下さったことに感謝を申し上げます．また Shanna McGoff 夫人には，素晴らしい努力と根気で原稿をタイプして頂いたことに謝意を表します．

Herbert Hellerman
Thomas F. Conroy

訳 者 序 文

計算機システムの性能を論ずるとき，オペレーティング・システムは明らかに決定的な要因の一つである．現在は計算機技術の重要な転期に当っているといってよく，たとえば従来の大規模集中型から分散化の方向に進みつつある．オペレーティング・システムもこれに従って変ぼうして行くと考えられるが，しばらくの間は本書が中心的に記述している IBM 360, 370 から基本的に大きく離れて行くことはないであろう．本書はこれらの計算機のオペレーティング・システムの動作をもとに計算機システムの性能を論じており，その取扱いは直観的，シミュレーション的，数学的という 3 つの観点からなされている．また，これらの取扱いを初心者が理解するための概念，定義および数学的基礎の説明も前半で与えられている．

オペレーティング・システムの解説を行う著書は数多くあるが，それらによってオペレーティング・システムの機能を動的に理解するのにはかなりの困難さが感じられることが多い．本書は計算機システムの性能をオペレーティング・システムの動的なふるまいを通じて考察しているため，性能評価ということだけでなく，オペレーティング・システムの理解に極めて有効である．したがって，本書は大学学部あるいは大学院の教科書として有用であるばかりでなく，実務家の参考書としても推奨できるものである．

翻訳は筆者の研究室の田畑孝一助教授，阿草清滋助手，久保正敏助手の 3 氏にお願いし，筆者はそれらの査読，修正および調整の仕事を負った．上記諸氏には大変多忙な中で寸暇をさいて翻訳，校正を行って頂いたことを，ここで深く謝意を表する次第である．筆者はちょうど教室主任という役目をする年度に当り，教室諸雑務，学生就職などの多忙な業務にまぎれ，本書における自分の役割りを充分果せたか危んでいる次第である．

大 野　豊

目　次

第 1 章

計算機システムの性能評価

計算機システムの評価には，次の事項の考察が含まれる：

1. 性能（大ざっぱには，システムの速さやソリース利用を表わす量として定義される）
2. コスト
3. ユーザの便利さ
4. 信頼性

おそらく，最近の情報処理システムの最も基本的な特質は，その一般性にあるだろう．この一般性を理論的に基礎づけたのは，A. M. Turing (1936) である．市販されているほとんどの計算機は，それらが入出力することができる文字セットや，利用可能な記憶容量についてだけ条件をつければ（記憶装置の技術や速さは基本的には条件とならない），論理的には互いに同等であの同じ広いることを示すことができる．なぜなら，これらの計算機は，同じクラスの，**計算可能**な諸関数を計算することができるからである．計算機の評価における最も困難な問題の多くは，計算機の機能がこのように極端に一般的であることによって生じてくる．すなわち，どんなシステムもそれが持つ機能によってしか評価できず，したがって機能が広がるほど，そのシステムの"良さ"を特徴付けることはますますむずかしくなるからである．

コストが上がれば性能も向上するという事実は，よく理解できることではあるけれども，両者の間の量的関係が全く明らかでない場合が多い．ユーザの便利さと性能との関係もまた定量化するのは非常にむずかしい．ほとんどの望ましい特性と同じように，ユーザの便利さは，ふつう，システムのコストと性能の両者を犠牲にはするが，当然ながら人間の生産性を向上させ，このことは，両者の犠牲を補って余りあるのである．ユーザにとって便利であるためには，

物理的システムをユーザ向きに変更させて，物理的な構造には必ずしも適していない間接的な方法でそのリソースを使えるようにする必要があり，このことが，システムの速さを犠牲にする理由の一つである．また，ユーザの便利さは，もともと個々のプログラマがやる仕事を自動的に行なうシステム・プログラムによって，少なくともある程度は達成される．もっとも，考えうるあらゆるユーザ・プログラムのそれぞれ個別な要求を最適な方法で満たしてやるような用具を設計するのはむずかしい．しかしながら，慎重にやれば，システム・プログラムはたいていのプログラマが書いたそれに対応するプログラムと，少なくとも同じ位の仕事を行うことがしばしば可能である．

ユーザの便利さに対するシステムの特性を**量的に**はかる尺度があれば非常に望ましいが，あいにく，手に入れることはほとんどできない．たとえばFORTRAN言語は，人間の生産性に大きく貢献していることはほとんど疑いないにもかかわらず，アセンブリ言語のコーディングよりどれだけ優れているかという価値尺度を，われわれは現在でさえ持っていないのである．この種のギャップが，システム評価という学問の発展の主な妨げとなっている．

1969年頃から，計算機システムの性能評価に関する文献が出始めた．これはある程度は，データ処理産業の成熟を反映している．いまや，匹敵する機能を持つ，いくつかの利用可能な装置やソフトウェア要素からの選択を賢明に行なうことに，ますます注意が払われるようになった．システムの経済化をはかろうとする強い誘因があり，それに対し少なからぬ技術的努力がなされているが，いまだにシステム性能評価の学問を生み出してはいない．克服しなければならないいくつかの障害には次のものがある．システムの性能について仕事をしている人たちの間で，性能を表わすさまざまな量の定義に非常な差があること．また，性能を表わすさまざまな量をお互いに関係づけるような解析的方法が不足していること．さらに，あるシステムから別のシステムへ，性能を表わす量を換算する一般的な方法がないことがあげられる．言い換えれば，われわれは性能評価の理論を持っていないのである．このような理論に向かっての進歩はおそいが，これは，定義や解析における技術的問題のためだけでなく，性能，ユーザの便利さ，信頼性，コストといったものが業務上重要な意味を持つためでもある．その結果，性能評価に関する多くの道具や結果が，それらを開発した企業に専有されたままになっている．

この章の残りでは，計算機システムの性能に関する問題点のいくつかに簡単に触れるが，その多くについては本書の後の方でさらに詳しい議論を行なう．

1-1　評価を行なう動機

計算機システムの評価は，それぞれ独自の見通しを持ったいくつかのクラスの人々から関心を持たれている．その大ざっぱなクラス分けは，次のようになる：

1. ハードウェア，ソフトウェアの設計者および製作者
2. システム設計の監理者と，そのシステム・プログラム作成者
3. ユーザ(応用プログラム作成者と解析者)

それぞれの人々の守備範囲は，上に並べた順に狭くなる．システムの設計者は，システムの可能な使用方法の全般的な状況によって影響される要因を扱わねばならない．特定の設備の管理者は，多くのユーザと関係するが，限定された環境だけを考えればよい．個々のユーザは，彼の1つのプログラムか，あるいはプログラムの小さな集りに関心がある．関心の範囲がより限定されるにしたがって，性能の目標や尺度はより正確に記述できる．しかし，リソースを扱ったり，特定の目標に到達するための自由度もまたより少なくなる．このように，設計者の選択の自由は相対的に大きいが，メーカから引渡されたシステムで仕事をする設備管理者は，そのシステムにとって使用可能なリソースに限定される．しかしながら最近の多くの計算機システムは，そのオペレーティング・システム (OS) の中に“オープン”パラメータを持っており，プログラム格納用に使う装置やスケジューリング方法などのような，システム効率にとって重要な選択は，おのおののシステムで作業負荷特性にあわせて決定することができるようになっている．このような決定を賢明に行なうことが，性能を良くするうえでの主要な要因となる．現在ではこの“**調整**(tuning)”の大部分は人手により行なわれているが，将来のオペレーティング・システムは“適応性”を持つであろう．すなわち，使用可能なリソースを探し，活動状況をモニタし，リソースの割りつけを自動的に行なうようになるだろう．しかしそうなった場合でも，ユーザ(あるいは設備管理者)は，自分達の性能の目標をシステムに教えてやらねばならないであろう．

個々のユーザは，リソースの割付けに非常な束縛をうけてはいるが，性能に重大な影響を及ぼすことが多い．われわれが最近経験した例では，繰返しの方法を変えると，あるプログラムの実行時間が (50倍も) 改善されたが，この比は，大型機と小型機の間での平均演算速度の比に匹敵するのである．経験豊富な人は大てい，同じような例を示すことができる．プログラムの品質は実際上

大きな幅を持っていることは事実である．これは問題解決を行なう際の人の技量に大きな差があることを反映しており，そしてプログラミングというのは，まさにその問題解決を行なうことなのである．

1-2　リソースの集合としてのシステム

現在の計算機システムは，リソースの集合としてとらえるのがよい．これらのリソースと，その現在典型的に実現されたものは次のようになる：

1．記憶装置
- （a）レジスタとバッファ；論理回路，ロジック・アレイ，高速コア，薄膜記憶アレイ
- （b）主記憶装置；薄膜コア，トランジスタ集積回路
- （c）補助記憶装置；コア，ドラム，ディスク，遅延線，磁気テープ

2．処理装置
- （a）算術論理とプログラム・制御のオペレーション
- （b）データ転送制御

3．情報変換装置
- （a）カード・リーダ，カード・パンチャ，テープ・リーダ，テープ・パンチャ
- （b）タイプ・ライタ，プリンタ，ディスプレイ

4．システム・プログラム
- （a）サブルーチン・ライブラリ
- （b）言語プロセサ
- （c）リソース管理ユーティリティ

工学技術の経済性を考えた結果，現在までずっと，システムは，スピードやコストがさまざまに異った部分構造——特に記憶装置——から構成されてきた．技術は変化しつつあるけれども，将来においても，異った技術は，異ったコスト-スピード特性を生み出すだろうと考えるのは，過去のすべての経験からみて，誤った予測ではないと思われる．このことから，次のことが言える．すなわち，スピード-コスト比がどのように改良されたものであったとしても，(過去と同様に)将来のシステムは，いろいろな技術の混合物からできあがったものであろう．この階層の管理は依然としてシステムの性能の中心問題であろう．

リソースの割りつけをプログラムで行なうことができるため，割りつけは伝

統的に個々のユーザ・プログラムにまかされてきた．ユーザが実行中にすべてのリソースを使用できるような専用システムでは，リソース割りつけは，負担は大きいが正当な仕事である．しかし，共有システム(マルチプログラミング・システム) では，予測できない時点で複数のユーザが共用リソースに対して同時に要求を出すことがある．もし1人のユーザに共用リソースを与えすぎると，他のユーザの死活にかかわるので，注意深いリソース割りつけが必要となる．このようなリソース要求の衝突は，**システム**によって解決されねばならない．なぜなら，個々のユーザは，リソースの状態を知らず，他のユーザに対するリソース割りあての動機や責任を持っていないからである．こんなわけで，**スーパ・バイザ**と呼ばれるシステム・プログラムが大部分のリソースのスケジュール，割りつけを行なう．ユーザにとっては，このプログラムはハードウェアの一部と考えてもよく，ユーザは多くのオプションを使用できるが，性能にとって重要な因子はそのスーパバイザの構成の仕方によって決定される．このようなシステム・プログラムはたいていのプログラムによって広範囲に使用されるが，個々のユーザはこのプログラムがどのように使われるかについては詳しくは知らない．システムの評価を実際に行なうには，システム・プログラムの性能も考慮しなければならない．

最後に，システムのオペレータがシステムの性能の要素として重要な役割を演じていることを認識しよう．コンソールによる対話や，機械室での人手による操作 (磁気テープやディスクの装塡，プリント出力の仕分けなど)は，システム全体としての性能に重大な影響を与えるのである．

1-3　性能の尺度 [BA, GO, HE2, LU 参照]

性能の量的尺度は，以下に述べる項目が明確に規定されてはじめて意味がある．

1. システム構成(すべての装置とソフトウェア・パラメータ)
2. 作業負荷あるいはジョブの流れ
3. 測定尺度の定義

第1の項目は明白ではあるが，一般的に述べることはむずかしい．システム構成に関する情報は非常に多いが，ほとんどの性能の効果にはふつうその一部分しか寄与しない．作業負荷を明確に記述したり，外挿したりすることのむずかしさについてはすでに述べたが，この本では繰返し論ぜられる題目である (1-4 節参照)．今のところは，ジョブ・ストリーム中のジョブ i は，到着時刻

と，実行あるいは処理時間との対 (A_i, X_i) によって表わす．すべての A_i が省略されておれば，すべてのジョブは，時刻 0 に到着する．

性能の尺度の定義はシステムの目標について，特に人間の期待に関して，注意深い考慮がなされている．非常によく使われる 2 つの尺度は，**スループット**と**応答時間**である．

スループットは，“バッチ”システムに対して一般に用いられる尺度で，1 分当たりのジョブ数，すなわち定められたジョブ・ストリーム中のジョブの数をそのストリームの全処理時間で割ったもので表わされる．

$$H = \frac{n}{T}$$

これは，全く単純なものにみえるが，実際には単純とはほど遠いものである．たとえば，マルチプログラミング・システムにおいては，ジョブの到着時刻とともに，ジョブの順序が非常に重要となることがある．またスループットは，各ジョブがいつ終了するかについては何も述べておらず，ただ要するに**最後の**ジョブがいつ完了するかを教えてくれるだけである．

ジョブ i の応答時間は，終了時刻と到着時刻との差 $(Q_i - A_i)$ で表わされる．平均応答時間は次の式で表わされる．

$$\bar{t}_e = \frac{1}{n} \sum_{i=1}^{n} (Q_i - A_i)$$

平均応答時間は平均化されたものであるので，しばしば重要となるようなある種の細目を隠してしまうに違いない．たとえば，次に示すような，2 つのシステム(共にオーバヘッドなし)により処理される，ジョブが 5 つのストリームを

		ジョブ 1	2	3	4	5
システム A:	実行時間	2	20	2	8	3
	応答時間	2	22	24	32	35
	スループット = 5/35 = 0.143；平均応答時間 = 115/5 = 23					
システム B:	実行時間	2	20	2	8	3
	応答時間	6	35	8	23	13
	スループット = 5/35 = 0.143；平均応答時間 = 85/5 = 17					

考えてみよう．2つのシステムは，ジョブをサービスする順序が異なっている．スループットは，両システムで同じであるが，平均応答時間は，システム B の方がずっとよい．また，システム A より B の方が応答時間が早いジョブは3つであるが，その反対のジョブは2つだけであることに注意しよう．

スループットは単に全ストリームを処理するのに要する時間の尺度を与えてくれるだけであり，個々のジョブが完了する時刻や，それらのジョブの完了順序については何も述べていない．このようにジョブの完了順序には敏感ではないので，応答が速い環境(たとえばタイムシェアリング)におけるスループットの重要性は，バッチ・システムにおけるそれよりも小さくなってしまう．したがって平均応答時間の方が，希望する目的にとってのよりよい尺度となる．しかし，応答時間が非常に重要な場合においてさえ，スループットはやはり興味深い．なぜなら，スループットは，ふつうシステム・リソースの効率的な使用と密接な相互関係があるからである．したがって，スループットが，ユーザの直接的な満足に対して，応答時間ほどには重要な関連をもたない時でさえも，スループットはコストに直接関係がある．

スループットに影響を与える重要な一般的要因としては，次のものがあげられる．

1. コンポーネント(システム構成要素)のスピード
2. 作業負荷
3. コンポーネントの使用において許されるオーバラップ(同時使用)
4. 作業負荷にリソースを割り当てるスケジューリングの方法

最初の要因は明白である．作業負荷は，スループットや，他のいかなる有用な性能尺度に対しても深い影響を持っている．この事実は非常に重要なので，この本の中でしばしば反復されるであろう．しかしながら，いろいろな尺度の**感度**は，同一の作業負荷の集合に関して全く同一である必要はない．このことを示すために，作業負荷が W_j の時のシステム i のスループットを $H_i(W_j)$ としよう．2つのシステムが，2つの異なる作業負荷に関して比較されると仮定しよう．4つのスループットの値が，いろいろな方法で比較されうるが，そのうち，(1) スループット比　(2) スループット差　の2つの方法を引きあいに出そう．作業負荷に対する感度を調べるためにこれら2つの方法を比較してみよう：

$$比の差 = \frac{H_2(W_2)}{H_1(W_2)} - \frac{H_2(W_1)}{H_1(W_1)} = a - b$$

$$差の比 = \frac{H_2(W_2) - H_1(W_2)}{H_2(W_1) - H_1(W_1)} = \frac{H_1(W_2)}{H_1(W_1)} \cdot \frac{a-1}{b-1}$$

上記の最初の方が，2番目のものより小さいのが普通である．すなわち，2つのシステムのスループット比は，スループット差よりも，作業負荷に対してより独立となる傾向にある．

オーバラップ，すなわち2つ以上のリソースの同時作動を許すことは，スループットに大きな影響を与えうる．簡単な解析によって，スループットの上限値が求まる．n-リソースのシステムと，逐次的に（すなわち，オーバラップなしに）実行されて，リソース j を t_j 秒要求するジョブとを考えよう．ジョブが逐時的に走る時間は，当然次のようになる：

$$T_S = \sum_{j=1}^{p} t_j$$

可能な最善のオーバラップ条件（すべての作業負荷に対して可能とは限らないが）を仮定すれば，全リソースは同時に使用され，ジョブの実行時間は次の式で得られる：

$$T_V = \mathrm{Max}(t_j)$$

したがって，オーバラップなしのシステムに対する改善率は，次のようになる：

$$\frac{T_S}{T_V} = \frac{\sum_{j=1}^{p} t_j}{\mathrm{Max}(t_j)}$$

もちろん，各 t_j は，システムとジョブの両方によって決まる．しかし，この改善率が最大となるのは，各 t_j がすべて等しく，したがって和の各項が最大値1をとる場合である．それゆえ

$$\mathrm{Max}\left(\frac{T_S}{T_V}\right) = p$$

これによって，次の原理が成立する：

可能な最も都合のよい作業負荷に対して可能な限りリソースの使用をオーバラップすれば，同じリソースを持つオーバラップなしのシステムに対するスループットの改善率は，リソースの数に等しい．

たとえば，2つのI/Oチャネルと1つのCPUから成るシステム（すなわち，

リソースが3つ）は，オーバラップなしのシステムより，**最高**3倍まで速くなりうる．たいていのジョブは順序に依存しているということが理想的なオーバラップを妨げるが，それを別にして他にもオーバラップを妨げる要因がある．動作をオーバラップさせて時間をかせぐには，ふつう空間の共用を（たとえばバッファ用に）必要とし，これはプログラムとデータの構成に影響を与える．このことが，逐次処理に関しては，オーバラップ・システムとすることに反する方向に作用しがちである．

1-4　作業負荷の選択：ベンチマーク
［BU 2, DO, DR 2, GO, JO, RO, SR, WA 参照］

いずれの性能の尺度も，取扱うべき作業負荷の詳細な規定を必要とする．この章の導入部分で示したように，たいていの計算機システムの汎用性というのは，計算機が非常に多様なジョブを処理できるということを意味している．実用的に考えると，**ベンチマーク**すなわちシステムに期待される仕事を代表するものとしては，多くの可能なジョブのうちからほんの少し選ぶことが必要となる．ベンチマークを選ぶには，どんな規準を使えばよいだろうか．多くのうちから3つの規準を挙げれば，次のものがある：

1. 最も頻繁に実行されるジョブ
2. システム稼動時間の大部分の原因となるようなジョブ
3. 応答時間（終了時間）に関する要求が，システムの機能を行なううえで最もきびしい条件となるようなジョブ

この1．と2．の違いは，特別な注意をはらうに足る事項である．たとえば，典型的な実行時間で走るジョブの**個数**の80％を占めるのは短かいジョブであり，これらはシステム**稼動時間**の（たとえば）30％にしか寄与せず，稼動時間の残り70％は，実行時間が長いわずか3つのジョブに帰因するという場合が起りうる．明らかに，このようなシステムの作業負荷の代表は，“大部分のジョブ”を代表するいくつかの短かいジョブと，“大部分の稼動時間”を代表するいくつかの長いジョブとを含まねばならない．

われわれは評価の仕事においては，同じシステムの現在の性能の改善として，あるいは新しいシステム構成を決定するために，**将来**の性能に関する決定を行なう際の助けとなる知識をさがすことが多い．いかにして，将来の作業負荷の代表を選べばよいのだろうか．たいてのい場合，過去に走ったジョブを選んで代表としている．これはしばしば全く正当である．しかし他の場合には，

疑わしいことがある．たとえば，バッチ処理のみを行なっていた設備が，バッチとタイムシェアリングの混合処理を行なうよう変換されるような場合である．確かに，システムが将来出くわすであろう作業負荷を識別することに，あらゆる努力がはらわれるべきである．もしこれが不可能であれば，少なくとも作業負荷の限界を知るべきである．

作業負荷選択が，たいていのシステム評価研究における最も扱いにくい事項であるというのは誇張ではない．比較しようとしているシステム構成が，機能的には互いに非常に似かよっている場合には，上で議論した考察が適用される．（機械命令語セットやプログラミング言語が異なる場合と同様）機能上の特徴が異なる場合をも含めて比較しようとすれば，作業負荷を定式化したり，結果を解釈したりする際に，より大きな注意を払わねばならない．たとえば，比較しようとするシステムが異った言語を用いている場合には，作業負荷となるプログラムを書く人たちのプログラミングの熟練度やスタイルは，現実に，結果に対して捕え難い影響を与えるのである．

数年前から，**合成作業負荷**(synthetic workloads)すなわちリソース使用の特性が作業負荷の特性を反映するように特別に作られた人工的なジョブに対して次第に関心が持たれるようになってきた．(BU2, SR, および4-2節参照)

1-5　リソース・エクササイザ：カーネル[CA, DO, DR2, SM 参照]

リソース・エクササイザは，選び出したシステム・リソースを，注意深く管理されたやり方で動作させてみるために設計されたプログラムである．この方法は，科学的方法論の立場からみて著しい利点を持っている，システム分析者の中には，この方法が，システムの比較に役立つ意味ある答を与え得ると感じている人がいる．しかしながら，この方法は，各エクササイザそれ自身が不完全な尺度であるという欠点を持っており，エクササイザを働かせて得られた性能を，興味の対象である現実の特定の問題を同じシステムに与えた場合の性能に関係づけることは，ふつう困難である．

それにもかかわらず，少くとも1つのタイプのエクササイザ，すなわち“カーネル”が，評価研究，特に科学計算においてしばしば使用されてきた．この場合には，プログラム・セグメント――通常これは，よく使われるプログラムの，1つあるいは複数の内部ループを表わすものであるが――が，興味の対象の計算機に対してプログラムされる．その計算機の命令タイミングを用いて，そのカーネルを実行する時間が計算される．カーネル法を適用するために使われるデータは，たとえばミックス法(1-6参照)に比べると比較的手に入れやす

いし，また，CPU 設計の初期段階において，種々提案されている命令語セット，アドレッシング，およびインデクシングの特徴を評価するために特に有効である．この目的のためには有用であるけれども，カーネル法は入出力の動作を全く無視しており，したがって，完全なシステム性能解析手法ではない．このカーネル法の考えを拡張して I/O 動作を含ませることは可能であろうが，今までほとんど行なわれていないように思われる．しかし，I/O の動作を含んだカーネルに似たものは使われてきている．その例としては 6 章で述べるような I/O と計算のオーバラップを調べるための，作業負荷の簡単なモデルがある．

1-6 命令ミックス [DR 2, G 1 参照]

スループットや応答時間(あるいはカーネル実行時間)に比べて，作業負荷に特別依存するということがより少ないような性能の尺度をさがすのは当然のことである．それは特に，システムが汎用性を持つと，実行される可能性のある作業負荷の数が非常に大きくなるからである．このような尺度をさがす 1 つの方法は，**命令ミックス**と呼ばれる平均化手法であり，計算機の CPU 部分を評価する際にこの方法が少なからず有効であることが証明されてきた．この方法では，1 つあるいは複数のプログラムが(通常，インタプリティブに)走り，与えられた型の命令が実行されるたびに，その命令の度数カウンタの内容が 1 増やされる．そして，各命令の型の実行時間によって，度数表に重みがつけられる．この重みつきの和は，その計算機の“平均命令実行時間”である(**図 1-6.1** 参照)．そして，他のシステムの平均命令実行時間を得るために，そのシステムの各命令実行時間と共に，同じ度数分布が使われる．そこで 2 つのシステムの性能のある種の相対的尺度を得るため，この 2 つの平均値を比較することができる．それぞれのシステムの命令語セットが非常に似ている場合には，この方法は最も有効である．

命令ミックス法は，いくつかの不利な点と限界を持っている．この方法は，命令語セットが似かよった計算機に対してその CPU の性能の有用な尺度を与えてくれるだけであり，入出力の性能を無視している．トレースを行なうことは，計算機の使用時間からみて非常に不経済である．すなわちトレースされるプログラムは，トレースなしのプログラムよりおそらく 50 倍も遅く走るであろう．命令のオーバラップをいろいろな度合で行なう大きなシステムにとっては，個々の命令の度数というものは，オーバラップによる多くの効果を反映していない．結局，度数分布は，それを得ようとしてトレースされるプログラムに

命令あるいは演算	度　数[3]	時間/演算	積
浮動小数点の加減算[1]	0.095	15	1.425
浮動小数点の乗算[1]	0.056	10	0.560
浮動小数点の除算[1]	0.020	10	0.200
ロード/ストア	0.285	4	1.140
インデクシング	0.225	0	0
条件付分岐	0.132	4	0.528
雑[2]	0.187	4	0.748
	1.000		
平均命令実行時間			4.601

注： 1. これらの命令は実行時間が一定していない．時間の重みとしては，平均時間がしばしば使われる．
2. このカテゴリは，シフト，イミーディエイト，およびロジカルの命令タイプを含む．
3. この表は，IBM　7090の"科学計算"タイプのオブジェクト・プログラムをトレースして作成された．

図 1-6.1　命令ミックス計算の実例

依存したものである；．いくつかの種類のプログラム(たとえばFORTRANコンパイラと，一般によく実行されるいくつかの応用プログラム)をトレースして，おのおののプログラムに対し別々のテーブルを作ることは，非常に賢明である．この他にもまた，難解な問題が生じる可能性がある；たとえば，システム$S1$が気のきいたコンパイラを持っていると，$S1$で生成された目的コードは，$S2$のコンパイラが生成した目的コードとは異なった度数分布を持っているかもしれない．

命令ミックス法は度数分析という強力な統計的手法の一つの実例である．このような方法の主な限界は，度数の計数が，データの中に入っている順序関係を含むことができないことである．多くの場合，特に小さな計算機においては，順序に関する情報はそれほど重要ではない．しかし，CPU内に階層的な記憶構造を持っているIBMシステム/370　モデル155，165などのような大きなシステムでは，性能は，命令やデータ参照の度数だけでなく，それらの順序に依存するであろう．このような場合には，命令ミックス法はほとんど適用できない．

1-7 システム活動の測定と推察［AP, DR2, KO, LU, STA1参照］

次に示す3種類の道具が，性能の研究に利用できる：

1. 解 析
2. シミュレーション
3. 測 定

この節では，測定に重点をおく．測定は特別な設備，たとえば，ハードウェアの活動状況をモニタする装置や，たとえば最近の多くのオペレーティング・システムに組込まれているイベント記録プログラムなどのソフトウェアによって行なわれる．

ハードウェア・モニタは，“**可搬性**(portability)”という大きな利点を持つ．たとえばいくつかのイベント・カウンタから成る同一のハードウェア・モニタを，多くの種類の計算機に，それらが実行しているプログラミング・システムにかかわりなく物理的に接続することができる．ふつうカウンタは，CPU，チャネルおよび周辺装置が動作状態，遊びあるいはいろいろな組み合わせのオーバラップ状態となっている時間を記録するために接続される．ハードウェア・モニタは設備の各部分の活動状況を記録するのには何にでも向いているが，ハードウェア・モニタによって記録した活動状況と，その測定期間中に作業負荷を構成していたプログラムとを互いに関連づけることは，しばしば非常に困難である．

ソフトウェア・モニタは，ふつう，CPUへの“割込み”が起るたびに呼出されて実行されるプログラムである．たいていのシステムでは，割込みは重要なイベントが起るたびに生起される．なぜなら．割込みはリソースやプログラム部分の動作完了をシステムに知らせる主な手段だからである．したがって割込みを傍受することにより，ソフトウェア・モニタ・プログラムは，重要なイベントが生じるたびに（システムの時計やインタバル・タイマにアクセスして），その時刻を記録することができる．注意深くやれば，この記録の仕事は，システムに対してほんのわずかな負担をかけるだけですむ；これらの記録の（通常磁気テープ上のレコードからの）データ整理は，正規のユーザ・プログラムとして行なうことができる．このデータ解析の結果によって，各プログラムがどこで大部分の時間を費やしているかとか，装置やデータ・セットへのアクセス頻度などを表わす情報を得ることができる．この情報は，より頻繁にアクセスされる情報ほどより速い記憶装置におくように記憶装置の再わりあてを決める際には，非常に貴重なものである．このようにすると，ディスク・アーム

のような共通パス上で要求がかちあうことは最も少なくなる．この種のデータは，設置されたシステムを"**調整**(tuning)"する場合のみならず，システムの開発中に"パフォーマンス・バグ"や"ホット・スポット"(激しく活動する領域)を識別する場合にも，大きな価値がある．ソフトウェア・モニタのこの他の重要な使いみちは，システムとそのオプションをモデル化する性能シミュレータ・プログラムを動かすために必要なデータが，ソフトウェア・モニタによって得られることである．このようなシミュレータは，性能の調整，新規購入，システム設計などを行なう場合に，"･･･としたらどうなるか"という質問に答えるための非常に役立つ道具となり得る．

1-8　オペレーティング・システムのクラス

オペレーティング・システムは，器機やプログラム・リソースの使用をわりつけたりモニタしたりする責任を負ったプログラムの集まりである．この本の最後の数章では，性能の観点から，いくつかのオペレーティング・システムを検討する．われわれの目的は，これらのシステム例の膨大な詳細を伝えることではなく，これらの時宜にかなった例を用いて，重要な性能のカテゴリや論点を示すことである．

章や節の標題からわかるように，オペレーティング・システムの特徴として3つの異ったクラスが認められる：

1. マルチプログラミング
2. タイムシェアリング
3. 仮想記憶

ここでは，これらの短く簡単で非形式的な定義を与え，次いでこれらを組み合せて考えてみよう：

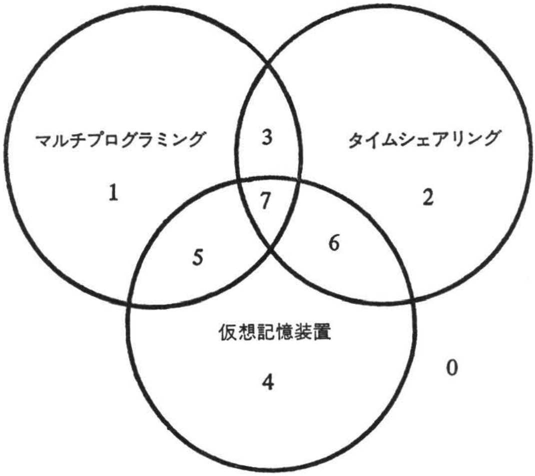
マルチプログラミング・システムは，いくつかのユーザ・プログラムが記憶装置，特に主記憶を空間的に共有するシステムである．通常，ユーザ・プログラム間で，リソースの時間的共有(タイムシェアリング)も行なわれる．

タイムシェアリング・システムは，複数のユーザが同時にそのシステムを使用できるように設計されており，各ユーザは，**対話的に**，すなわち人間一機械の直接的な対話が楽に行なえるような便利さと時間遅れをもって，プログラムの作成，デバグ，実行を行なうことができる．

仮想記憶システムとは，論理的な特性と速さが異なる2つ以上の記憶装置から成る階層的な記憶構造を持ち，それが，ユーザにとっては単一の論理的な

記憶装置として見えるように，自動的に管理されているシステムである．

鋭い読者なら，これらのカテゴリ間には何ら**本質的**な関連がないということを見出すこともっともである．厳密に論理的な意味においては，このことは事実であるけれども，最近の多くのシステムは，これらのカテゴリをいろいろに組み合わせたものとなっている．これを考えるには，ベン図（図1-8.1）が役に立つ．前述の3つのカテゴリは，互いに交差する3つの円で表わされている；カテゴリの可能な組合わせで，当然8つの領域が存在しうる．ベン図の下



タイプ	注釈と実際例
0	初期のシステムと最近の小型システム
1	最近のバッチ・システム（たとえばIBM　OS/360/370　MVT）
2	MIT/CTSS：いくつかの独立形タイムシェアリング・システム
3	1972年以前のほとんどのタイムシェアリング・システム (IBM TSO, GE Information Services network)
4	まれである．というのは，ほとんどの仮想記憶システムはマルチプログラミングも行なっている．
5	1972年に発表されたIBM VSシステム；初期のFerranti-ATLASシステム
6	まれである．というのは，ほとんどの仮想記憶システムはマルチプログラミングを行なっている．
7	GE/MIT MULTICS, Michigan大学のMTS, IBM TSS, IBM CP/67, IBM VMシステム

図 1-8.1　オペレーティング・システムのカテゴリとその組合せ

に，各領域の簡単な注釈と実際例を掲げておく．さらに詳しくは，この本の最後の部分に述べられている．

1-9 結　　言

計算機システムの性能評価は，完全な技術的学問としては未だ揺籃期にある．少くとも3つの広範なグループの人々が，評価の情報を必要としている：システム設計者，システム設備の管理者およびシステム・プログラマ，そして応用プログラマである．各グループは独自の要求を持っており，それらの要求を満たすために，それぞれ異なった程度の，リソースを選択したり制御したりする自由度を必要としている．

計算機システムが実行できるのは非常に広いクラスの問題であり，システムはスピードがいろいろなリソースから構成されており，また異なったジョブによるリソースの使い方に違いがあるので，いかなる性能の尺度も作業負荷の詳細な規定を含んでいなければならない．これは非常に難かしい処置である．という理由の1つは，ある作業負荷を与えたシステムの性能を他の作業負荷を与えて得た性能から推論するための，性能評価の理論がないことである．

作業負荷の詳細な規定を行なうための，将来有望な1つの方向は，作業負荷に対して，現実のジョブを要求することよりもむしろ，ジョブを確率的に表現することである．少数の統計的なパラメータを使って設計されたジョブを想像することは可能である．その場合には，ジョブ・ストリーム生成プログラムは，現実の作業負荷としてシステムにとりこまれるような合成ジョブ・ストリームを生成したり，あるいはまた，システム・シミュレータに与えるのに適したジョブ・ストリームの骨組みを生成したりするであろう．この手法は，今日現在シミュレータにおいてはよく使われているが，測定の研究分野においてはまだ広く使われてはいない．統計的，確率的な方法は，工学・社会複合システムの評価においてはすでに広く使われているが，将来，計算機システムが，もっと多くのユーザに，(タイムシェアリングや情報処理ユーティリティ・システムでのように) 予想がもっとむずかしい方法で使われるようになるにつれて，統計的，確率的な方法の重要性は，計算機システムにおいても確かにますます大きくなるであろう．こういうわけで，本書の最初の部分が，統計や確率の方法に関与しているのである．最後に，計算機システムほどに複雑な構造物の簡単で“一般的”な評価というようなものは，実際にはないのであるということを強調しなければならない；おのおのの性能の研究は，まずその目標を決め，つぎに特に作業負荷と性能の尺度を選ぶ際に，その目標を反映しなければならな

い．性能の結果は，それが測定，シミュレーション，解析(あるいはこれらの組合せ)のいずれによって得られたとしても，選択された作業負荷や性能尺度に関してのみ，意味があるのである．目標の定義が少しでも変われば，性能の値のみならずシステム間における性能値の順位もまた変わってしまうのが当然であるということに注意しよう．

演 習 問 題

1. コンピュータの一般性が，性能研究における根本的なむずかしさの原因であるのはなぜかを説明せよ．

2. 次の状況は適応性があるかないかを識別せよ．
   (a) ユーザは彼のプログラムが必要とする主記憶装置の量を宣言しなければならない．
   (b) 現代の典型的なストリングまたはリスト処理言語（たとえばSNOBOL, LIST)における記憶空間の管理．
   (c) ジョブはユーザが宣言した優先順位の順序で処理される．
   (d) 実行時間が最も短いと判断されたジョブが最初に処理される．
   (e) プログラム P は，B の方が主記憶が多いという以外は相等しいシステム A と B の上で実行される．このテストの間，システムは P だけを実行する．B の方が実行が早い．

3. スループットと応答時間との間の重要な相異について簡潔に論じ，それぞれがより適切な性能尺度となるような例をあげよ．(より詳しい議論は4-5節参照)

4. 2つのジョブ・ストリームを使って，2つのシステムが比較されるものとする．実行時間を測定して次の表が得られた．
   (a) 各システムについて各ストリームのスループットを計算せよ．
   (b) それぞれのストリームについてスループットの比を計算せよ．比の差を計算せよ．

実 行 時 間

ストリーム	ジョブ数	システム 1	システム 2
1	25	2.50	5.00
2	10	0.67	1.00

（c）　スループットの差の比を計算せよ．(b)と結果を比較せよ．

5．次のような3つの課金番号の記録が与えられている．

課金番号	ジョブ数	平均実行時間	ジョブ数の%	実行時間の%
0001	300	30秒		
0002	50	20分		
0003	150	2分		

（a）　上の表の空欄を埋めよ．

（b）　次の質問を論ぜよ．システムの使用の大部分の原因となるのはどの課金番号か．大部分のジョブを処理したのはどれか？

6．タイムシェアリング・システムの主要な特徴を読者自身のことばで述べよ．この型のシステムに特有な特徴を強調せよ．

7．仮想記憶システムの本質を簡潔に定義せよ．記憶階層（たとえば，磁心記憶装置とディスク）の性質に関する読者の基本的知識にもとずき，仮想記憶システムに**タイムシェアリング**あるいは**マルチプログラミング**のどちらか1つ**だけ**を含む場合の(性能の見地から見た)相対的な重要性について考えよ．

参　考　文　献

1. [AP] Apple, C.T.: The Program Monitor–A Device for Program Performance Measurement, *Proc. ACM 20th Natl. Conf.*, August, 1965, pp. 66-75.
2. [AR] Aaron, J.D.: Real Time Systems in Perspective, *IBM Systems Journal*, vol. 6, no. 1, 1967.
3. [BA] Bagley, P.R.: Two Think Pieces: Item 2: Establishing a Measure of Capability of a Data Processing System, *Comm. ACM*, vol. 3, no. 1, p. 1, January, 1960.
4. [BU1] Bucholz, W.: A Selected Bibliography on Computer System Performance Evaluation, *IEEE Computer Group News*, vol. 2, no. 8, March, 1969.
5. [BU2] Bucholz, W.: A Synthetic Job for Measuring System Performance, *IBM Systems Journal*, vol. 8, no. 4, pp. 309-318, 1969.
6. [CA] Calingaert, P.: System Performance Evaluation: Survey and Appraisal, *Comm. ACM*, vol. 10, no. 4, pp. 12-18, April, 1967.
7. [DO] Dopping, O.: Test Problems Used for Evaluation of Computers, *BIT*, vol. 2, no. 4, pp. 197-202, 1962.
8. [DR1] Drummond, M.E.: A Perspective on System Evaluation, *IBM Systems Journal*, vol. 8, no. 4, p. 252, 1969.
9. [DR2] Drummond, M.E.: "Evaluation and Measurement Techniques for Digital Computer Systems," Prentice-Hall, New York, 1973.
10. [GI] Gibson, J.C.: The Bigson Mix, *IBM Technical Report* TROO. 2043, June, 1970.

11. [GO] Gosden, J.A. and R.L. Sisson: Standardized Comparisons of Computer Performance, *Information Processing 1962* (Proc. IFIP Congress 62), pp. 57-61.
12. [HE1] Hellerman, H.: "Digital Computer System Principles," (2nd Ed) McGraw-Hill Book Company, New York, 1973.
13. [HE2] Hellerman, H.: Some Principles of Time Sharing Scheduler Strategies, *IBM Systems Journal*, vol. 8, no. 2, 1969.
14. [HEN] Hendrie, G.C. and R.W. Sonnenfeldt: Evaluating Control Computers, *ISA J.*, vol. 10, no. 8, pp. 73-78, August, 1963.
15. [IB1] IBM Corp.: *IBM Systems Journal*, vol. 5, no. 3, 1966. (Contains six articles on the design of teleprocessing systems.)
16. [IB2] IBM Corp.: *IBM Systems Journal*, vol. 8, no. 4, 1969. (This issue contains several articles summarizing the state-of-the-art, as of date of article, on performance measurement and evaluation.)
17. [JO] Joslin, E.O.: "Computer Selection," Addison-Wesley Publishing Company, Reading, Mass., 1968.
18. [KI] Kittel, H. and P. Mertens: Einige Quantitative Untersuchungen zur Groessendegression von Datenverarbeitungsanlagen. (On the economy of scale of data processing systems.) *Elektronische Datenverarbeitung*, vol. 7, no. 6, pp. 255-262, December, 1965.
19. [KO] Kolence, K.W.: A Software View of Measurement Tools, *Datamation*, January, 1971.
20. [LU] Lucas, H.C.: Performance Evaluation and Monitoring, *Computing Surveys*, vol. 3, no. 3, pp. 79-91, September, 1971.
21. [MA] Martin, J.: "Design of Real Time Computer Systems," Prentice-Hall Book Company, New York, 1967.
22. [RO] Rosin, R.F.: Determining a Computing Center Environment, *Comm. ACM*, vol. 8, no. 7, pp. 463-468, 1965.
23. [SO] Solomon, M.B. Jr.: Economies of Scale and IBM System/360, *Comm. ACM*, vol. 10, no. 2, pp. 435-440, June, 1966.
24. [SM] Smith, J.M.: A Review and Comparison of Certain Methods of Computer Performance Evaluation, *The Computer Bulletin*, pp. 13-18, May, 1968.
25. [STA1] Stanley, W.I.: Measurement of System Operational Statistics, *IBM Systems Journal*, vol. 8, no. 4, pp. 299-308, 1969.
26. [STA2] Statland, N.: Methods of Evaluating Computer System Performance, *Computers and Automation*, vol. 12, no. 2, pp. 18-23, 1964.
27. [STE] Stevens, D.F.: System Evaluation on the Control Data 6600, *Proc. IFIP Congress 1968*, pp. 542-547.
28. [STI] Stimler, S.: "Real Time Data Processing Systems," McGraw-Hill Book Company, New York, 1969.
29. [WA] Walter, E.S. and V.L. Wallace: Further Analysis of a Computing Center Environment, *Comm. ACM*, vol. 10, no. 5, pp. 266-272, 1967.
30. [WH] White, P.: Relative Effects of Central Processor and Input-Output Speeds Upon Thruput on the Large Computer, *Comm. ACM*, vol. 7, no. 12, December, 1964.

第2章

統計学の原理

統計学(statistics)ということばは，“**国家**(state)”ということばと同じ語源を持っている．この事実によって，17世紀のイギリスで行政上および商業上の必要から近代的な形で起ったこの学問の起源をかいま見ることができる．たとえば,住民の平均余命を予言するための技術とともに,人間の寿命についての信頼しうるよく組織されているデータがなければ，保険会社は機能できない．

統計学は，データの正確な記述に関する数学の一分野である．ほとんどの近代的システムを合理的に設計する際の重大な初期の段階は，データの収集と組織化であるか，あるいは少なくともそうあるべきである．なぜならば，システムの入力と期待される出力の特性が明確に述べられていないと，システムをうまく規定することはできないからである．そのような記述は，見通しのあるユーザおよび彼らのプログラムのふるまいや，その結果としての実際のシステムでの応答などの数多くの観察から導き出されねばならない．われわれの関心は計算機システムにあるけれども，他の種類のシステムにおいても，統計学はますます必須な要素となってきている．たとえば，選挙における投票の統計学は，候補者や政党が次のシステム(次の選挙選)をいかに設定するかを決めるときに助けとなる．

計算機システムの場合には，聡明に設計しようとすれば，いろいろなレベルの細目についてのおびただしいデータが必要となる．ハードウェア・レベルでは，命令語セットやアドレッシングの方法の選択は，プログラムを走らせて観測された，統計上から見たリソースに対する要求，特に記憶装置に対する要求を反映しなければならない．仮想記憶を含む階層記憶構造の設計もまた，そのような情報をもとに行なわれてきた．すべての場合に，普遍的な目標というのは，繰返し生じる活動のパターンを識別しそれらが設計過程においてそれ相応に重要視されるようにすることである．

今日では，ほとんどの統計計算や，いくつかのデータ収集まで，計算機によって行なわれている．確かにそれ自身の動作に関係する限りは，計算機システムは，ユーザによるリソースへの要求やそれらの要求へのシステム自身の応答を簡単にモニタすることができる．このように，計算機システムは，統計学の見事な適用場所である．この適用は，2つの異なる時期に行なわれる．

1. **設計時：**　設計の“パラメータの値”たとえば，特定の命令語タイプ，記憶装置の大きさや速さなどを決定するために．
2. **実行時：**　システム動作中の“実際のふるまい”を記録するために．それはリソース割り当てのスケジューリング・アルゴリズムへの入力として役立つ．そのようなアルゴリズムは，よいサービスを表わすある量を最適化するように設計されている．

上記の考えは，すでにいくらかは使われているが，われわれが統計的設計の可能性に手をつけはじめたばかりであることは，ほとんど疑いがない．ますます，システムの世界は，統計の世界となってきているのである！

統計的手法への熱意をくじくおそれはあるが，統計的手法が誤りを犯すわなと機会にみちているということを強調しなければならない．最も一般的な誤りは (1) 不完全なデータ収集，(2) 統計理論の公式と結果を引出してきたモデルの仮定にそむくような状況にそれらを適用することといった，統計理論の公式と結果の見境いのない使用，の2種類である．知らず知らずこれら2種類の誤りをおかす機会はたくさんある．さらに，経済上および社会上の大きな問題がかかわってくるので，ある望みの結果を得るために，故意に誤った考えが持ちこまれることも時にはある．このことにより，統計的手法は，一般大衆にとって不評となりがちであったけれども，改善の方向は明白である．すなわち可能な限り組込み検査を行なうだけでなく，統計的手法とその適用の方法をよりよく理解し，より明白に解説することである．

統計学の主な着想は簡単ではあるが深遠である：たとえば理性的で細心な行動をとるための入力や刺激といったような，ある意図した目的のためにデータの意味を引出すため，大量のデータの集りを組織し，正確にそれらを記述することである．**正確に記述する**ということばによって，われわれは，数字を使用するということに気づく；統計学の大部分は，それよりずっと多くのデータ値の集合を特徴づけることができるような数の小集合を，賢明に定義するための，情報圧縮の技術として理解されるであろう．この理由により，われわれは，平均値，モード，標準偏差などといった，いくつかの**分布の中心的特性を表わ**

す尺度(measures of central tendency)を定義することから始める．次いで，いくつかの度数分布関数と，それらの性質のいくつかを議論する．最後に，最小二乗法の意味でデータに“最もよくあてはまる”多項式曲線を見つけることによってそのデータを特徴づける方法を述べてこの章を締めくくる．おそらく，最もよく見られるのは，データにあてはまる多項式が直線の場合である．これは線型回帰と呼ばれ，データを補間したり，データの傾向を見つけたりするのに用いられる．

2-1 モード(最頻値)，平均値，標準偏差

X という名前がついた，N 個の数値から成るデータ・セットあるいはデータ・ベクトルがあるとする．以後，次の数値を例として用いる：

$$X = 1,\ 3,\ 14.5,\ 16.4,\ 18,\ 7,\ 14.6,\ 2,\ 0.5,\ 6$$

X **の範囲**(range)は，その最大値と最小値の対である†

$$R_X = \mathrm{Min}(X),\ \ \mathrm{Max}(X) = 0.5,\ \ 18 \tag{1}$$

最も簡単(単峰的)な場合，**モード**(mode)は，最も出現回数が多い X の値である．同じ最大出現回数を持つ値が2つ以上出現すれば，データは**多峰的**(multimodal) であると言われ，モードは，最大出現回数を持つすべての異なった値のベクトルとなる．

算術平均値(arithmetic mean)は，簡単に X の**平均値**(mean, average)とも呼ばれるが，X の要素の和を要素の個数で割ったものとして定義される．

$$M = M_X = \bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i = \frac{1}{N}\sum X = 1/10(83) = 8.3 \tag{2a}$$

この式の左には，平均値を表わすのによく使われるいくつかの記号が並んでいる；2つ以上あるデータ集合のうちどの集合を考えているかを規定するために添字が必要となる場合以外は，本書では添字なしの M を使うことにする．総和を表わしている一番右側の形(上式の第5項)は，添字に関する細目が除かれているが，この形も，便利な略記形として使うことにする；したがって，添字は，そのすべての値をとるものと仮定される．

X のすべての要素に対し同一の定数 c が取り込まれたならば，平均値は，次に示すような影響をうける［証明は，式(2a)から簡単に導びかれる］：

† 範囲は[$\mathrm{Max}(X) - \mathrm{Min}(X)$]であると定義する著者もある．

$$M_{X+c} = M_X + c \text{ および } M_{X-c} = M_X - c \tag{2b}$$

$$M_{cX} = cM_X \text{ および } M_{X/c} = M_X/c \tag{2c}$$

平均値は，最も広く使われている，データの中心的傾向を表わす尺度であるが，データの値の**ばらつき**についての情報は，何ら与えていない．たとえば，2つのデータ集合

$$X = \ \ 2,\ \ 3, 59, 24$$

$$Y = 22, 25, 19, 22$$

は，同一の平均値(22)を持っており，したがってこの平均値では，Y が20あたりの値のまわりに密集しているように見え，一方 X はそうではないということを示すことができない．もし，データがある値のまわりに密集していると考えられるならば，あるいはもしわれわれが，単に X の各値の h からの偏差を調べたいと考える場合でさえも，X の各値と h との差 D をとり，そして全データ集合の h からの偏差を表わす尺度として，これらの差の，ある簡単な関数を作り出すということは，道理にかなっている．われわれがとりあげる測定用の関数の形としては，差の二乗の和の形を選ぶのが合理的である．なぜなら，差を二乗することによって，和をとる時に正の差と負の差が打ち消しあわないということが保障されるからである．この和は，データ集合が大きくなるにつれて，必ずしも値のばらつきが大きくなるという理由によってではなく，単にデータの個数が多くなるという理由によって，より大きくなる傾向にある．このことを避けるため，われわれはこの和をデータの個数で割り，**差の二乗平均**をもってわれわれが用いる尺度とする．

$$M_D{}^2 = \frac{1}{N}\sum_{i=1}^{N} D_i{}^2 = \frac{1}{N}\sum_{i=1}^{N}(X_i - h)^2 = \frac{1}{N}\sum(X-h)^2 \tag{3}$$

さてこんどは，次のことが問われねばならない：差をとるときに参照すべき h の値は，どのように選べばよいであろうか．もちろん答は1つではないが，1つの合理的な基準は，差の二乗平均($M_D{}^2$)が**最小**になるように h を選ぶということである．この条件を使えば，h についての公式は，変数 h に関して導関数を0とおき関数の極小値を求めるという微分学の手法を適用することによって導びかれる．方程式(3)にこれを施せば，求まった h は，データの平均値に等しいことがわかる．すなわち

差をとるための<u>共通参照値</u>がデータの平均値である時には，他に考えられるいかなる参照値を用いた場合よりも，平均二乗差は小さい．

この，平均値からの差をとって得られる，平均二乗差の最小値は，**分散**と呼ばれ，その平方根は**標準偏差** S_X と呼ばれる．

$$S^2 = S_X{}^2 = \frac{1}{N}\sum_{i=1}^{N}(X-M)^2 \quad あるいは \quad S = \sqrt{1/N\sum(X-M)^2} \qquad (4)$$

どんなデータ・セットが与えられても，標準偏差を計算するには，式 (4) を使うことができる．この計算を行なうには，2つのプログラム・ループが必要である．一つは，平均値を得るのに必要な和を求めるためのもので，他方は，平均値からの差の二乗の和を求めるためのものである．計算量の価値ある節約を行なおうとすれば，まず式 (4) にちょっとした代数を施せばよい．二乗の項を展開し，M と M^2 を総和からくくり出し，式(2)を使えば

$$S^2 = \left(\frac{1}{N}\sum X^2\right) - M_X{}^2 = \left(\frac{1}{N}\sum X^2\right) - \left(\frac{1}{N}\sum X\right)^2 \qquad (5a)$$

となる．右辺の2番目の形は，効率改善が約束されていることを示している．というのは，X と X^2 の総和は，1つのプログラム・ループで求めることができるからである．

平均値と標準偏差の物理単位は，データ値のそれと同じである．しかし，両者の値が，どちらもデータのうちのどれかの値に一致する必要がないということは，注目に値する．

もし，X のすべての値にある定数 c が導入された場合には，式 (5a) から証明されるように，分散は次のごとく影響をうける：

$$S^2{}_{X\pm c} = S_X{}^2 \qquad (5b)$$

$$S^2{}_{X/c} = \frac{1}{c^2}S_X{}^2 \qquad (5c)$$

標準偏差を表わす便利な方法は，平均値に対する標準偏差の比である**変動係数**(coefficient of variation)で表わすことである：

$$CV = \frac{S}{M} \qquad (6)$$

X と Y との2つに分かれたデータ・ベクトルを結合して1つの集合 X, Y にすることがしばしば必要となる．X の平均値および Y の平均値と，結合されたデータの平均値との間の関係はどうなっているか．同じように，結合されたデータの標準偏差は，それぞれの標準偏差によってどのように表わされる

か．平均値に対する方程式を導出し，次いで標準偏差に対する結果を述べることにしよう．まず，結合データの平均値は，定義によって次のようになる：

$$M_{X,Y}=\frac{\sum X+\sum Y}{N_X+N_Y} \tag{7}$$

もちろん

$$M_X=\frac{1}{N_X}\sum X \qquad M_Y=\frac{1}{N_Y}\sum Y \tag{8}$$

である．式(8)を総和について解き，式(7)に代入すれば，次式が得られる：

$$M_{X,Y}=\frac{N_X M_X+N_Y M_Y}{N_X+N_Y} \tag{9}$$

結合平均値に関する基本法則は，2つ以上のデータ集合に対しても拡張される．以下に述べる，興味ある特別な場合は，式(9)から導びかれる．

平均値が等しい($M_X=M_Y$)場合：

$$M_{X,Y}=M_X=M_Y \tag{10}$$

データ数が等しい($N_X=N_Y$)場合：

$$M_{X,Y}=\frac{M_X+M_Y}{2}=\text{平均の平均} \tag{11}$$

標準偏差の二乗を結合するために，まずその定義を示す．

$$S^2{}_{X,Y}=\frac{\sum(X-M_{X,Y})^2+\sum(Y-M_{X,Y})^2}{N_X+N_Y} \tag{12}$$

上で平均値を求めた時と同じような手順(代数的にはもっと複雑であるが)により，標準偏差の二乗に関し，次式が求まる：

$$S^2{}_{X,Y}=\frac{N_X S_X{}^2+N_Y S_Y{}^2}{N_X+N_Y}+\frac{N_X N_Y(M_X-M_Y)^2}{(N_X+N_Y)^2} \tag{13}$$

いくつかの特別な場合については，この一般式から導びかれる．

平均値が等しい($M_X=M_Y$)場合

$$S^2{}_{X,Y}=\frac{N_X S_X{}^2+N_Y S_Y{}^2}{N_X+N_Y} \tag{14}$$

この式は，平均値に関する式(9)に似ていることに注意しよう．

データ数が等しい$(N_X = N_Y)$場合：

$$S^2{}_{X,Y} = \frac{S_X{}^2 + S_Y{}^2}{2} + \frac{(M_X - M_Y)^2}{4} \tag{15}$$

例：　次にあげるのは，ある計算機センタで入手できる2, 3, 4月の統計である．

月	ジョブ数	平均ジョブ実行時間	標準偏差
2月	2000	10分	4分
3月	3000	12分	5分
4月	3000	15分	6分

（a）　2, 3月2カ月間の，ジョブ実行時間の平均値，標準偏差を求めよ．

（b）　2〜4月3カ月間の，ジョブ実行時間の平均値，標準偏差を求めよ．

解：　（a）　式(9)を使えば

$$M_{F,M} = \frac{(2000)(10) + (3000)(12)}{2000 + 3000} = 11.2\text{分}$$

式(13)を使えば

$$S^2{}_{F,M} = \frac{(2000)(4)^2 + (3000)(5)^2}{2000 + 3000} + \frac{(2000)(3000)(10 - 12)^2}{(2000 + 3000)^2}$$

$$S_{F,M} = 4.7\text{分}$$

（b）　読者への演習問題：［**ヒント：**2〜3月をまとめて1つの“仮の月”と考え，(a)で求まった値を用いよ］

2-2　標準化変数

データの値の形を正規化することにより，データやデータの関数の比較がしばしば非常に簡単になり，ある種のデータの差異に，より焦点をあてることができる．データ X の各要素に，次に示す変換を施すことにより，**標準化変数**と呼ばれる新しいベクトル Z が得られる：

$$Z = \frac{X - M}{S} \tag{1}$$

このように，Z は，X とその平均値との差を標準偏差で割ったものである．X,

M, S は同じ物理単位を持つので，Z は元なし(物理単位なし)である．

標準化変数には，次に示すさらに2つの重要な性質がある：

$$M_Z = 0 \tag{2}$$

$$S_Z = 1 \tag{3}$$

これらは，前に述べた平均値と分散の定義に式(1)を代入することによって，簡単に求まる．

2-3　平均値の型

算術平均値は，多くの種類の平均値の中でも，最も広く使われている．他の種類の平均値が，しばしば定義される：

幾何平均値：　$$G = \sqrt[N]{X_1, X_2, \cdots, X_N} \tag{1}$$

調和平均値：　$$H = \frac{N}{\sum 1/X} = \frac{1}{1/N \sum 1/X} \tag{2}$$

平方平均値：　$$\mathrm{rms} = \sqrt{M_{X^2}} = \sqrt{1/N \sum X^2} \tag{3}$$

rms 値は，次の式に示されるように，標準偏差と算術平均値に密接な関係がある：

$$\mathrm{rms} = \sqrt{S_X{}^2 + M_X{}^2} \tag{4}$$

2-4　度数によって分類されたデータ

分析のためにデータを組織化するための1つの非常に重要な方法は，**度数の手法**を用いることである．これは，次に示す手続き(**図2-4.1**)によって行なうことができる．

1. X のすべての相異なる(重複しない)値のリストを作り，値が増加する順に並べる．この N' 個の値のベクトルを X' と呼ぶ．すなわち

$$X'_{i+1} > X_i' \qquad i = 1, 2, \cdots, N'-1$$

2. データ・ベクトル X における，各 X_i' の出現頻度(度数)を数え，その数を f_i とする．

度数分布は，2つのベクトル(X'; f)の組から成り，1番目のベクトルはすべての相異なるデータ値を増加順に並べたものを与え，2番目のベクトルは，

原データ

$X = 1, 4, 12, 9, 3, 5, 4, 12, 9, 3, 4, 12, 1, 9, 3, 12$

度数によって分類されたデータ

$X' =$	1	3	4	5	9	12
$f =$	2	3	3	1	3	4
$\frac{f}{N} =$	$\frac{2}{16}$	$\frac{3}{16}$	$\frac{3}{16}$	$\frac{1}{16}$	$\frac{3}{16}$	$\frac{4}{16}$

$N = \sum_{i=1}^{N'} f_i$

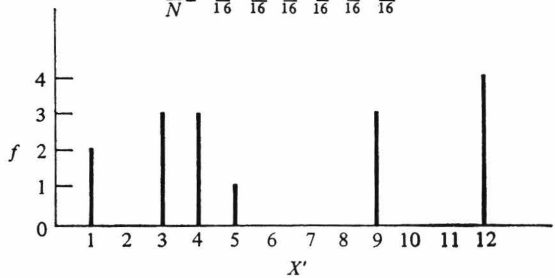


図 2-4.1　度数によって分類されたデータ

おのおのの値がデータ中に出現する回数を与えている．頻度の関数を求めるためのもう一つの方法は，まずデータ X を，非減少の順序で**ソート**することである．そうすると，繰り返し現われる値は隣接するので，繰り返しの回数を数えて f を求め，次いで値の繰り返しを取り除き X' を求めることは，簡単となる．度数分布の形で組織化されたデータは，**分類されている**と言われる．

f のすべての値の和は，もとのデータ X の値の個数を数えあげることになるので，

$$\sum_{i=1}^{N'} f_i = \sum f = N \tag{1}$$

$$\sum_{i=1}^{N'} \frac{f_i}{N} = \frac{1}{N} \sum f = 1 \tag{2}$$

である．

ベクトル f/N は**相対度数ベクトル**と呼ばれ，ベクトル $(X'; f/N)$ で形成される関数は，**相対度数分布**と呼ばれる．図 2-4.1 は，説明用の例である．

相対累積度数分布は，f/N を f/N の部分和のベクトルで置き換えることにより得られる．したがって，その i 成分 C_i は，X_i に“等しいかあるいはより小さな”値のデータの個数の，全個数に対する割合を示す．

$$C_i = \frac{1}{N}\sum_{j=1}^{i} f_j \qquad i = 1,\ 2,\ \cdots,\ N' \tag{3a}$$

式(3a)から得られるもう一つの便利な形は

$$C_i = C_{i-1} + \frac{f_i}{N} \qquad i = 1,\ 2,\ \cdots,\ N' \tag{3b}$$

ここで，$C_0 = 0$ である．式(2)を使えば，$C_{N'} = 1$ が得られる．

もしデータがすでに分類されておれば，データの中心的特性を表わすいろいろな尺度を計算する労力は意味ありげに軽減され得る．

分類されたデータに適用できるいくつかの式を与えよう．X' は順序づけられている(ソートされている)ので，

$$\textbf{範　囲:}\quad R_X = X_1',\ X'_{N'} \tag{4}$$

である．

おのおのの X_i' の値は X において f_i 回現われているので，そのような値の総和は f_iX_i' であり，したがって

$$\text{平均値:}\ M = \frac{1}{N}\sum_{i=1}^{N'} f_iX_i' = \frac{\sum fX'}{\sum f} \tag{5}$$

$$\text{標準偏差(2乗):}\ S^2 = \frac{1}{N}\sum_{i=1}^{N'} f_i(X_i' - M)^2 = \left(\frac{1}{N}\sum fX'^2\right) - M^2 \tag{6}$$

$$\text{モード(最頻値):}\ \mathrm{Mode} = X'_{\underset{j}{\mathrm{Max}}}(f_j) \tag{7}$$

モードは，度数分布の頂点における X' の値(複数個になる場合もある)である．

度数分類についてさらに詳細に考えを進める前に，この方法の利益，不利益は何かを問うことは，もっともなことと思われる．一つの基本的な不利益は，データを再編成する過程で，データの元の順序が失なわれる（もちろん，コピーを別に作っておかない場合のことであるが）ことである．元の順序が重要であるかどうかは別の問題でその答はある環境に依存する．

度数分類の主な利益は，データを保持するのに要する記憶容量の減少である．極端な例として，データが，たまたま同一の値をとる1000個の数値から成る場合をとりあげてみよう．ベクトル X を格納するためには，1000個の記憶セルが必要である；分類された形では，わずか2つのセルが必要となるにすぎない――1つはデータの値，1つは度数(1000)を記憶するためのものである．

実際にはたいていの場合，データのすべての現実の値について度数を求める必要がなく，単にデータ値のある**範囲**のみの度数を数えれば十分である．たと

原データ

$X = 0, .15, .20, .11, .77, .24, .70, .23, .28, .52, .10, .15, .75, .88, 1.0$

バケットの構成：　$R_1 = 0; R_2 = 1.0; \Delta = .1; m = 10$

バケット番号 →	1	2	3	4	5	6	7	8	9	10
L	0	.1	.2	.3	.4	.5	.6	.7	.8	.9
U	.1	.2	.3	.4	.5	.6	.7	.8	.9	1.0
f	1	4	4	0	0	1	0	3	1	1

図 2-4.2　バケット(クラス)を使った度数による分類

えば，各 X_i の値が $0 \leq X \leq 1.00$ の範囲にあるものと見込まれており，それらの値の度数としては，たとえば 0.1 きざみの区間にはいる値の度数でもいとわないという場合を仮定しよう．おのおののバケットが，可能な X の値(**図 2-4.2**)に対する区間（クラスの区間）に対応するような 10 コのバケットを思いうかべるとわかりやすい．おのおののバケット，たとえばバケット i には，次の 3 つの数値が結びつけられる：

$L_i =$ バケットの区間の下限
$U_i =$ バケットの区間の上限
$f_i =$ バケットの区間に入るデータ値の度数

任意の値 X_j は次に示すあるバケット i の範囲に“入る”であろう．

$$L_i \leq X_j < U_i \tag{8}$$

バケット i に数えられるのは，その下限値以上で上限値までで上限値を**含まない**範囲に入るすべての値であるということに注意しよう．U_i に等しい値はバケット i に入らないものと見なす理由は，重複して数えるのを避けるためである．というのは，U_i はまた L_{i+1} でもある，すなわち，各バケットの上限値は，次のバケットの下限値であるからである (**図 2-4.2**)．しかしながら，最後のバケットはその上限値を含んでいると定義されている．なぜならその次のバケットはなく，それゆえ数えあげが重複する危険がないからである．

一旦バケットの上下限値が決められると，データ・ベクトル X が処理される時に度数 f_i がどのように決められるかを見るのは，概念的には簡単である．このことについて，簡単にもう少し述べてみよう．さしあたり，**バケット設定**の過程を一般的な形で要約しよう：

1. データの**範囲**を決定する.

$$R_1 = \text{Min } X \qquad R_2 = \text{Max } X \tag{9}$$

もし R_1, R_2 の一方あるいは両方の値が未知で，しかもデータをざっと調べることを別途行なうのが不可能な場合には，R_1 と R_2 を推定してやり，1つは R_1 以下，1つは R_2 以上のあふれの度数を記録するための2つの余分のバケットを作ればよい.

2. バケットの区間 $\varDelta$ を決める(すべてのバケットに対し同一と仮定する).
3. したがって，バケットの総数は次式で与えられる.

$$m = \lceil \frac{(R_2 - R_1)}{\varDelta} \qquad \text{ここで} \lceil \text{は，等しいか，より大きくて最も近い整数をあらわす.} \tag{10}$$

4. 各バケットの上下限値は次式で求まる.

$$L_1 = R_1 \tag{11}$$

$$L_i = L_{i-1} + \varDelta = R_1 + (i-1)\varDelta \qquad i = 2, 3, \cdots, m$$

$$U_i = \text{Min}(R_2, (L_i + \varDelta)) \tag{12}$$

ここに示したバケット設定過程では，ユーザはデータ値の範囲とバケットの区間を規定することを要求されている——バケット設定における他のすべての数値はこれらから計算される．実際には，バケット設定時にユーザは範囲を知らないかもしれず，その場合にはデータを調べて範囲を知ったりあるいはデータ値に対して抱くある“感じ”により範囲を見積ったりするであろう．ある場合には，ユーザはバケットの区間よりもバケットの個数 (m) を直接指定したいであろう．その時，区間は次のように決められる：

$$\varDelta = \frac{R_2 - R_1}{m} \tag{13}$$

その後で，上下限値が上記のステップ **4.** によって計算される.

バケット設定過程は，データの特性についての大ざっぱな知識を多少必要とはするけれど，データ値の個数には無関係であるというまい特徴を持つということは，注意を払うに値する.

バケットの区間がすでに定義されており，すべてのバケットの度数が最初0にセットされていると仮定する．実際にデータ X の度数分布を作成する2つの方法を考えてみよう．**探索法**(search method)においては，各データの値 X_k を使って L の値(各バケットの下限値)と比較し，バケットの L の値が X_k よ

り小さいか等しくなるまで探索が進められる．見つかったバケットに対応する f の値が，1 増やされる．記号で書けば

$$j \leftarrow \operatorname*{Max}_{i}(L_i \leq X_k) \tag{14a}$$

$$f_j \leftarrow f_j + 1 \tag{14b}$$

インデクス変換法(index transformation method) はもう1つの方法である．しかしこれは各バケットの幅が等しい場合にしか適用できない．この一般的な場合には，この方法によればバケットの範囲を調べることがまったく不要となり，したがって計算時間が節約される．そのかわりに，各データの値はバケットを指す整数のインデクスに対応付けされる；そしてこのバケットの度数が1増やされる．もっと正確に言うと，データ X に対する度数ベクトルを作成するために，まずすべての度数を0に初期化する ($f=0$)．次に，おのおのデータ X_k に対し，バケット・インデクス i を次のように計算する．すなわち，m と，$(X_k\text{-}R_1)/\Delta$ より小さいか等しくて最も近い整数に1を加えた数とのうち小さい方を i とする．

$$i = \mathrm{Min}\left[\mathrm{m},\ \left\{1 + \lfloor \left(\frac{X_k - R_1}{\Delta}\right)\right\}\right] \tag{15a}$$

そして，バケット i の度数を更新する．

$$f_i \leftarrow f_i + 1 \tag{15b}$$

バケットの考え方は，度数以外の関数に対してもまた使われる．たとえば，各バケットは，そのバケットの範囲にあるデータ値を累積するのに使われてもよい．こうすれば，そのバケットの区間における平均値を計算することが簡単になる．

ある種の慣習や用語が，度数関数を図示したり表示するのによく用いられる．**ヒストグラム**(図 2-4.3)は，次のような性質を持つ図表である：

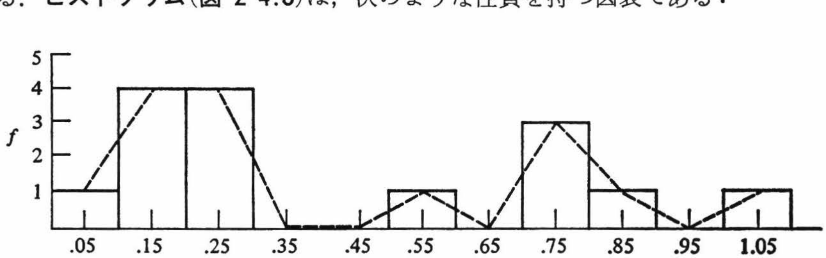


図 2-4.3　図 2-4.2のデータに対するヒストグラム

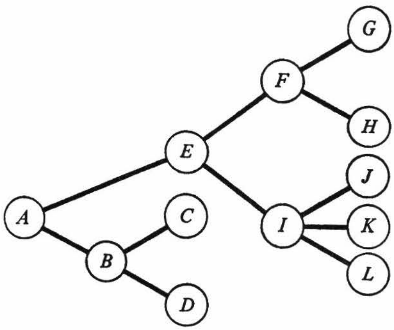
1. 水平軸(X' または X)は，各バケットの範囲の中央の値で目盛られている．
2. 各バケットの度数は，バケットの区間に等しい幅を持った矩形で図示される；矩形の X 軸上の底辺の中心は，バケットの中央値の位置に置かれる．
3. 矩形の**面積**は，度数に比例している．範囲が等しいバケットでは，すべての矩形の幅は同一で，高さは単に度数(あるいは相対度数)の値になる．ヒストグラムの矩形の頂上の中心点を次々と直線で結んで得られる折れ線は，**度数多角形**と呼ばれる．

累積度数を表示するために同じような矩形の図が使われることがある．[多角形で表わした図は**オジーブ図**(ogive plot)と呼ばれる．]垂直座標はしたがって，バケットの範囲の上限値より値が小さいデータの個数の全データ個数に対する割合を与えている．

例題　図 2-4.4 に示すような木構造が与えられているものとする．この構造は，機械の組立てやその一部分の組立て，さらにその一部分の組立て等に必要な部品を示す際に見られる．各ノードは，データへのディスク・アドレス(ポインタ)および木構造における後続ノードへのポインタを表わしている．この構造は，ノード・ラベルを探索するプログラムによって使われる；プログラムは，与えられたラベルに一致する構造を捜す．おのおののノードを通過するたびにアクセスが1回あるものと仮定すると，任意のノードに到達するのに必要なアクセス回数は，図表から簡単に決定され，ベクトル A として記録される．さて，100回の探索の後に，各ノードが要求された回数はベクトル F として記録されているものとしよう．アクセス回数に関して，次にあげる値を求めよ：

1. 平均値
2. メディアン(2-6 節参照)
3. 標準偏差
4. 分散係数
5. 度数分布
6. 相対累積度数分布

この問題の解を，図 2-4.4 に示す．



ノード	A	B	C	D	E	F	G	H	I	J	K	L
A（アクセス回数）	1	2	3	3	2	3	4	4	3	4	4	4
F（度数）	0	10	5	15	2	3	10	15	5	10	20	5

平均値

$$M_A = \frac{\Sigma FA}{\Sigma F} = \frac{348}{100} = 3.48$$　［式(5)より］

メディアン

A′（異なる値を順に並べたもの）→	1	2	3	4
F →	0	12	28	60
相対累積度数	0	.12	.4	1.0

A′,Fより，中央値（メディアン）は4である

標準偏差

$$N = \Sigma F = 100$$

$$S = \sqrt{1/N\, \Sigma F(A' - M_A)^2} = .6697$$　［式(6)より］

変動係数

$$CV = \frac{S}{M_A} = .20107$$

図2-4.4　木探索に関するいくつかの統計(例)

2-5　度数関数による観測のシミュレーション［IB, KN参照］

前に議論したように，度数関数は，多くの目的のために，多量のデータを表現する簡潔な方法である．しかし，そのような簡潔化の逆を行なうことがしばしば必要となる．すなわち，度数関数からデータの値を生成することである．たとえば，その作業負荷が1つまたはそれ以上の分布関数によって記述されて

いるような**システムをシミュレート**したい場合には，この要求が生じる．このようなシミュレーションでは，おのおのの動作単位を記述しているシミュレートされたデータ値が，シュミレータ・プログラムを通して送られることが要求される．たとえば，シュミレータは，おのおののシュミレートされたジョブの到着時刻，サービス時間を要求するであろう．これらの値は，あらかじめ行なった測定により得られた分布関数から求められねばならない．それでは，そのようなデータを生成するために度数関数を使う技法について考えてみよう．データ値 X に対して**相対累積度数**関数 $C(X)$ が使えるものと仮定しよう．X はどのような範囲の値でもとりうるが，$C(X)$ は相対度数であるので 0 から 1 の範囲の値をとる．プログラミングの見地からすれば，X と $C(X)$ とは**数字の組**の表であらわされるであろう．充分多くの値が生成されればそれらの相対累積度数分布が $C(X)$ に近づくような方法で，X の値のランダムな実例値を生成するために $C(X)$ を用いたいというのがわれわれの望みである．

この条件を満たすランダムな X の値を生成するために，まず，コールされるたびに，区間 $0 \leqq y \leqq 1$ にある乱数を発生する，標準的な乱数生成器が用意されているものとしよう．おのおのの乱数は，**一様な**分布を持っている；すなわち，その相対度数分布は水平な直線であり，その区間のすべての数値の出現確率は等しいことを示している．乱数生成器という題材は，相当に奥の深い問題である．† **図 2-5.1** は，FORTRAN による乱数生成器を示している．乱数生

```
C  RANDOM NUMBER GENERATOR
C  IX = 123(SAY)FOR FIRST CALL ON RANDU
C        USE IY FOR SUBSEQUENT CALLS
C  IY = INTEGER RANDOM NO. GENERATED(BETW. 0 AND 2**31)
C  YFL = FRACTION RANDOM NO. GENERATED(BETW. 0 AND 1)
   SUBROUTINE RANDU(IX, IY, YFL)
   IY = IX*65539
   IF(IY)5, 6, 6
 5 IY = IY + 2147483647 + 1
 6 YFL = IY
   YFL = YFL*.4656613E-9
   RETURN
   END
```

図 2-5.1 乱数生成器 [IBM 科学計算サブルーチン・パッケージ (H 20-0205)]

† 参考文献 8 の D. E. Knuth の論文参照．

成器によって生成された y の値は，次いで，以下に示すように合成データを生成するのに使われる(**図 2-5.2**)：

1. 一様乱数生成器をコールする．得られた値を y で表わす．
2. y を使い，一致する(あるいは補間される)$C(x)$の値を求める．
3. 対応する x の値が，必要な合成データの値である．

累積分布が解析的に与えられている場合には，上記と同様な過程は　以下に示すように行なうことができる：

例：　解析的に規定された指数分布を考えよう．

$$C(x) = y = 1 - e^{-0.04x}$$

この分布により合成データの値 x を生成するには，まず，x について解析的に解き**逆関数**を求めるのが便利である．

$$x = -25\ln(1-y)$$

したがって，上記の手続きをたどるのは簡単である．

1. 一様乱数生成器をコールする．たとえば　$y = 0.2$ が得られたとする．
2. 逆関数の y に 0.2 を代入し，x を求める

$$\begin{aligned} x &= -25\ln(1-0.2) \\ &= (25)(0.22314) = 5.5785 \end{aligned}$$

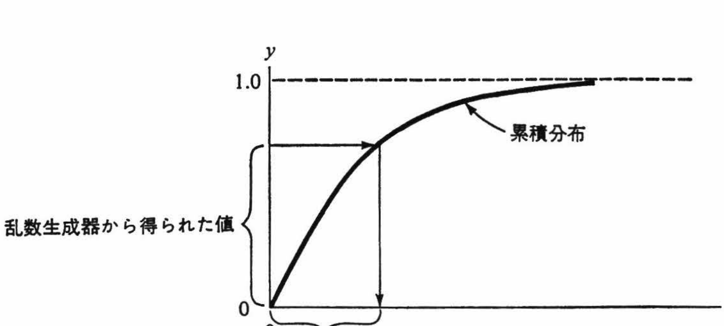


図 2-5.2　与えられた(累積)分布からの乱数の生成

2-6 メディアンと分位

X の**メディアン** (median) とは その値より大きいか等しい値の個数と，その値より小さいか等しい値の個数とが同じであるような値である．メディアンを見つけるには，X を増加する順序で（もっと正確には非減少の順序で）並べる；その結果のベクトルを X'' と呼ぶ．すなわち

$$X''_{i+1} \geq X''_i \qquad 1 \leq i \leq N-1 \tag{1}$$

したがって，X のメディアンは，N が奇数，偶数，両方の場合を含めて次の1つの式で定義される．

$$\text{メディアン} = \frac{X''_{1+\lfloor N/2} + X''_{\lfloor (N+1)/2}}{2}$$

ここで $\lfloor$ は，“等しいかより小さくて最も近い整数”を意味する．すなわち，

$$\text{メディアン} = X''_{(N+1)/2} \qquad N\text{ が奇数の場合} \tag{2a}$$

$$= \frac{X''_{N/2} + X''_{N/2+1}}{2} \qquad N\text{ が偶数の場合} \tag{2b}$$

言い換えれば，X の値のソートのあと，メディアンとは，データ個数が奇数の場合には中央の位置の値であり，偶数の場合には中央の位置の2つの値の平均値である．N が偶数の場合，メディアンの値が，X の要素である必要はない．

たとえば収入のデータなどでしばしば見られるように，データが少数の非常に大きな値を含んでいるような場合には，平均値よりもメディアンの方が，よりよいデータの記述子となることが多い．給料が＄8,000，＄9,000，＄11,000，＄12,000，＄85,000である5人の委員会を考えよう．給料の平均値は＄25000であり，メディアンは＄11,000である．明らかに，平均値よりもメディアンの方が，この委員会の給料構成をずっとよく表わしている．

さて，元のデータが，ある決められたバケットの方法を用いて，相対度数分布および相対累積度数分布に組織化されているものと仮定しよう．これについてのすべての情報は次に示すベクトルで表わされ得るが，このベクトルはある表における列であると思えばよい．

L	バケットの下限値
U	バケットの上限値
f/N	相対度数
C	相対累積度数

データの値 x が与えられており，それがどのバケットに属しているかとわれわれが質問されたらどうであろうか．答は，2-4節の式(14a)によって得られるバケット j である．

$$j = \operatorname*{Max}_i (L_i \leq x)$$

データの全個数のどのくらいの割合が，x より小さいか等しい値のデータであるかを知りたいならば，以後に述べるような道筋をたどる．この割合についての最初の大ざっぱな見積りは，C_{j-1} である．これは，バケット $j-1$ に対する相対累積度数である（$j-1=0$ に対しては，$C_0=0$ とする）．しかし，C_{j-1} ではほとんど常に過少見積りになる．なぜなら，実際にはこの値は，x を含むバケットより下のすべてのバケットに含まれるデータ値の総個数の割合を与えているからである．x を含むバケット j では，f_j 個のデータ値が数えられているが，この分布を見て，f_j 個のうちの正確に何個が，x より小さいか等しいのであるかを言うことはできない．しかし，線型補間によってこの数値を見積ることが可能であり，このことは，f_j 個の値がバケット j の範囲で一様に分散していることを仮定していることになる．この仮定の下で，x より小さいか等しい値の個数を与える補間値は，x 以下のバケット区間の割合を f_j 倍したものである．x より小さいか等しい値の割合 F に対する完全な公式は次のようになる：

$$j = \operatorname*{Max}_i (L_i \leq x) \tag{3a}$$

$$F = C_{j-1} + \frac{(x-L_j)f_j}{U_j - L_j} = C_{j-1} + \frac{x-L_j}{U_j-L_j}(C_j - C_{j-1}) \tag{3b}$$

データの値の相対的順位を割合あるいは小数 F で表わすかわりに，Q 個のバケットを使う方法においてデータの値が落ちこむバケットの番号を示す整数 q でもって，この順位情報を表わすことがよく行なわれ，またしばしば便利である．よく使われる Q の値は，$Q=4, 10, 100$ であり，それぞれ，**四分位**(quartile)，**十分位**(decile)，**パーセンタイル**(percentile)と呼ばれる．割合あるいは小数で表わされた順位 F を q に変換するには，

$$q = 1 + \lfloor FQ \tag{4}$$

説明のため，x が，式(3)によって $F=0.3$ となるような値であるとすれば，式(4)は，x の相対順位が，$Q=4, 10, 100$ の場合についてそれぞれ，第2の四分位，第4の十分位，第31のパーセンタイル，にあるということを表わしている．上に述べた方法は非常によく使われているけれども，順位を表わすため

にどんな Q の値を選ぶかは明らかに任意である．Q の値はいろいろ考えうるが，これらを総称して**分位**(quantile)という．

2-7　対称性の尺度

度数分布図は，しばしば次のような，ある種の全体的な特徴を表わしている．

1. **様相**(modality：頂点の数)
2. **歪度**(skewness：非対称の度合)

歪度は，**頂点**（モード）の一方の側の方が，他方の側よりも度数の値がゆっくり減少するという形であらわれる．ゆっくり減少する領域は，**長い方の尾**(longer tail)と呼ばれる．

歪度の，よく使われる2つの量的尺度は，平均値と中央値（モードあるいはメディアン）との間の差を含んだものである．これらは，Pearson **の第1および第2歪係数**(Pearson's first and second coefficients of skewness)と呼ばれる．

$$P_1 = (\text{平均値} - \text{モード})/\text{標準偏差} \qquad (1)$$

$$P_2 = 3(\text{平均値} - \text{メディアン})/\text{標準偏差} \qquad (2)$$

たとえば，右に歪んでいる，すなわち，頂点(モード)の左側よりも右側の方でゆっくり落ちるような分布は，平均値が右側にあり，したがって平均値はモードより値が大きいと予想される．よって P_1 は正であることがわかる．P_1 は，モードとメディアンが一致する時には0になり，モードが平均値より大きい時には負になる．

2-8　2変数の和の平均値と分散：相関

それぞれの値の個数が等しい2つのデータ・ベクトル X と Y を考える．この2つのベクトルの，対応する値を加える，すなわち $X+Y$ を作ったとしよう．これにより，独自の平均値と分散を持った新しいデータ・ベクトルができる．これらを M_{X+Y} および $S^2{}_{X+Y}$ と書くことにする．われわれはこれから，M_{X+Y} と $S^2{}_{X+Y}$ とを，M_X, M_Y, S_X, S_Y によって表わしてみよう．

平均値の場合はかなり簡単である：単にその場合に対して平均値の定義を適用し，少々の簡単な代数を施せば

$$M_{X+Y} = \frac{1}{N}\sum(X+Y)$$
$$= \frac{1}{N}\sum X + \frac{1}{N}\sum Y$$

したがって，和の分布の平均値は，平均値の和に等しいということになる.

$$M_{X+Y} = M_X + M_Y \tag{1}$$

標準化変数に対しては，平均値が0であるから

$$M_{Z_{X+Y}} = M_{Z_X} + M_{Z_Y} = 0 + 0 = 0 \tag{2}$$

2変数の和の分散は，導出するのがもう少し複雑である．分散についての一般的な定義から出発し，次いで変数を $X+Y$ で置きかえる.

$$S^2{}_{X+Y} = \frac{1}{N}\sum(X+Y-M_{X+Y})^2$$

M_{X+Y} に対して式(1)を使い，項を組み換える.

$$S^2{}_{X+Y} = \frac{1}{N}\sum[(X-M_X)+(Y-M_Y)]^2$$

差の部分はそのままにして，二乗を展開すれば

$$S^2{}_{X+Y} = \frac{1}{N}\sum[(X-M_X)^2+(Y-M_Y)^2+2(X-M_X)(Y-M_Y)]$$

和の中に総和を分配し，定義

$$S^2{}_X = \frac{1}{N}\sum(X-M_X)^2 \quad \text{および} \quad S^2{}_Y = \frac{1}{N}\sum(Y-M_Y)^2$$

を使い，さらに，X, Y の**共分散**という新しい定義

$$\mathrm{COV}(X,\ Y) = \frac{1}{N}\sum(X-M_X)(Y-M_Y) \tag{3}$$

を加えれば，和の分散を次のように表わすことができる.

$$S^2{}_{X+Y} = S^2{}_X + S^2{}_Y + 2\,\mathrm{COV}(X,\ Y) \tag{4}$$

共分散の定義，式(3)を使い，X と Y を標準化変数で置きかえれば

$$\mathrm{COV}(Z_X,\ Z_Y) = \frac{1}{N}\sum(Z_X-M_{Z_X})(Z_Y-M_{Z_Y}) \tag{5}$$

標準化変数の平均値は 0 であるから

$$\mathrm{COV}(Z_X,\ Z_Y) = \frac{\sum Z_X Z_Y}{N} \tag{6}$$

われわれはいまや，2変数 X, Y の共分散と，対応する標準化変数 Z_X, Z_Y の共分散との間の関係を導き出す準備ができた．標準化変数の定義

$$X - M_X = S_X Z_X \qquad \text{および} \qquad Y - M_Y = S_Y Z_Y \tag{7}$$

を思い出し，これらを式 (3) に代入し，共通な因数 S_X, S_Y をくくり出せば，次の式が得られる．

$$\mathrm{COV}(X,\ Y) = \frac{S_X S_Y \sum Z_X Z_Y}{N} = S_X S_Y\,\mathrm{COV}(Z_X,\ Z_Y) \tag{8}$$

ここで，最後の簡単化は，式(6)を使っている．

標準化変数の共分散は，**相関** r とよばれる．

$$r = \mathrm{COV}(Z_X,\ Z_Y) = \frac{\mathrm{COV}(X,\ Y)}{S_X S_Y} \tag{9}$$

モーメント積の公式(product-moment formula)と呼ばれる，r のもう 1 つの形は，次のようになる．

$$r = \frac{\sum(X - M_X)(Y - M_Y)}{N S_X S_Y} = \frac{\sum(X - M_X)(Y - M_Y)}{\sqrt{[\sum(X - M_X)^2][\sum(Y - M_Y)^2]}}$$

$$= \frac{\sum xy}{\sqrt{\sum x^2 \sum y^2}} \tag{10}$$

ここで，$x = X - M_X$，$y = Y - M_Y$ である．r の値は

$$-1 \leq r \leq 1$$

の範囲にある．r が 0 ならば，X と Y のデータは**相関がない**と言われる．

式(9)は，式(4)を次の形で表わすために使われることがある．

$$S^2{}_{X+Y} = S^2{}_X + S^2{}_Y + 2rS_X S_Y \tag{11}$$

この式は，幾何学における余弦定理――これは　c^2 (三角形の一辺の長さの二乗)と，他の二辺 a, b の二乗の和および c の対角 C の余弦とを関係づけている――に代数的に著しく似かよっている．

式(11)の特別な場合として，2つの変数が同じ分布を持ち，したがって同じ標準偏差を持っており，しかし相関がない$(r = 0)$ならば

$$S_{X+Y} = \sqrt{2}\ S_X \tag{12}$$

もっと一般的に，同じ標準偏差を持ったK個の相関がない分布の和に対して，和の変数の分布の標準偏差は，

$$S = \sqrt{K}\ S_X \tag{13}$$

となる．

2-9　最小二乗法による曲線のあてはめ：回帰［DA, DR 参照］

統計学の一つの目的は　多くの数値を少数の数値で表わすことであるということであった．これを行なう一つの方法は，データに，多項式のような解析的関数を“あてはめる”ことである．一旦これが行なわれると，少なくともいくつかの目的にとっては　データを表わすためには多項式のほんのわずかな係数を記録しておくだけでよい；擬似データの値は，必要に応じてこの多項式を使って計算される．この節では，データに多項式(直線を含む)をあてはめる一般的な方法を導出する．

それぞれ同じ個数のデータ要素を含む2つのデータ・ベクトルX, Yについてからまず始めよう．通常，データは，たとえばプログラムのバイト数とその実行時間などのように，ベクトルの間に何らかの関係があると考えられるようなデータである．各X_iの値と対応するY_iの値は，XY座標上の点の座標と考えることができる．このような点の集まりは，**相関図**と呼ばれる(**図2-9.1**).

相関図上に，データの点に“最も良く”あてはまるような直線を引きたいと仮定しよう．“最も良く”という概念は，直線が**すべての**点を通ることが明らかに不可能である(すべての点が直線上に乗るという非常にまれな場合を除いて)ということから生じている．それゆえ，どのように直線が引かれたとしても，一般には，各Yの値と，対応する直線上の縦座標との間には，いくらかの誤差があるだろう．そこでわれわれが“最も良く”あてはまるという言葉で意味するものを定義することに話を進めよう．データを推定するのに使われる**いかなる**直線の方程式も次の**形**をとるであろう．

$$Y_e = a_0 + a_1 X \tag{1}$$

ここでY_eは，推定された，Xに対応する縦座標である．今のところ，パラメ

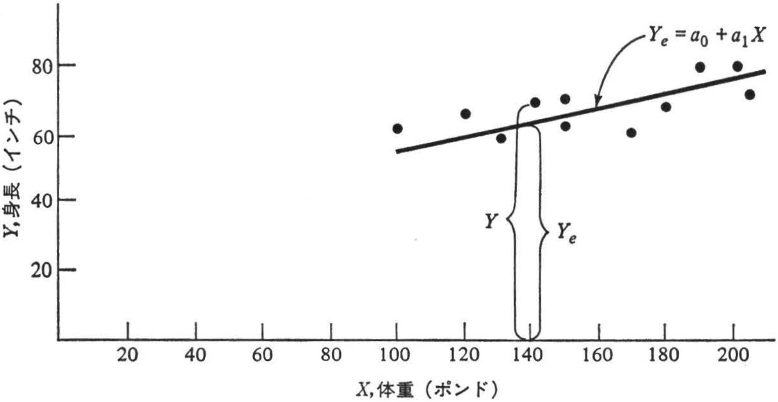


図 2-9.1 相関図と回帰直線

ータ a_0, a_1 の値は未知であるということに注意しよう．

直線とデータとの間の平均二乗誤差は，また**推定の標準誤差**(standard error of estimate) とも呼ばれ，データの値 Y と対応する近似直線上の Y_e の値との差を二乗したものの平均値で与えられる．

$$E^2 = \frac{1}{N}\sum(Y - Y_e)^2 \tag{2}$$

Y_e に対して式(1)を使えば

$$E^2 = \frac{1}{N}\sum(Y - a_0 - a_1X)^2 \tag{3}$$

そこでわれわれは，平均二乗誤差が最小となるような a_0, a_1 の値を見つけることにしよう．これは，微分学を使い，E^2 の（a_0 と a_1 についての）2つの偏微分を0とおくことによって可能となる．その結果 a_0, a_1 は，**正規方程式**(normal equation)と呼ばれる a_0, a_1 についての線型方程式

$$\sum Y = a_0N + a_1\sum X \tag{4}$$

$$\sum YX = a_0\sum X + a_1\sum X^2 \tag{5}$$

の解であることがわかる．この2つの線型方程式を a_0, a_1 について解けば：

$$a_0 = \frac{(\sum Y)(\sum X^2) - (\sum XY)(\sum X)}{D} \tag{6}$$

$$a_1 = \frac{\left(N \sum YX\right) - \left(\sum X\right)\left(\sum Y\right)}{D} \tag{7}$$

ここで，$D = \left(N \sum X^2\right) - \left(\sum X\right)\left(\sum X\right)$　(8)

a_0, a_1 に対する式(6)，(7)，(8)を式(1)に代入して決められた直線は，**回帰直線**(regression line)と呼ばれる．これは，すべての可能な直線のうちで，全データ点からの平均二乗誤差が最小であるような直線である．回帰直線は，多くの興味ある有用な性質を持っている．

定理 2-9.1　回帰直線は，X, Y の平均値の交点(M_X, M_Y)を通る．

証　明

1.　1番目の正規方程式(4)を N で割れば

$$M_Y = a_0 + a_1 M_X$$

2.　式(1)より，M_X, M_Y は回帰直線上の点である．

定理 2-9.2　推定の標準誤差，すなわちデータ Y と直線上の Y_e との平均二乗誤差は，次の式になる．

$$E^2 = \frac{1}{N} \sum (Y - Y_e)^2 = \frac{1}{N}\left(\sum Y^2 - a_0 \sum Y - a_1 \sum XY\right) \tag{9}$$

証　明

1.　定義により

$$E^2 = \frac{1}{N} \sum (Y - Y_e)^2 = \frac{1}{N} \sum (Y - a_0 - a_1 X)^2$$

2.　二乗を展開して

$$E^2 = \frac{1}{N}\Big[\sum Y(Y - a_0 - a_1 X) - a_0 \sum (Y - a_0 - a_1 X)$$
$$- a_1 \sum X(Y - a_0 - a_1 X)\Big]$$

3.　正規方程式(4)，(5)を代入すれば，2番目と3番目の総和は0となりしたがって式(9)となる．

定理 2-9.3　次の2式

$$x = X - M_X \tag{10}$$

$$y = Y - M_Y \tag{11}$$

で定義される新しい変数 x, y を使って，回帰直線は次のように書かれる．

$$y = a_1 x \qquad \text{または} \qquad Y - M_Y = a_1(X - M_X) \tag{12}$$

ただし

$$a_1 = \frac{\sum xy}{\sum x^2} \tag{13}$$

$$a_0 = M_Y - a_1 M_X \tag{14}$$

この証明は，上記の x と y の定義式を a_0 と a_1 の方程式に代入して代数処理を行なえば得られる．定理 2-9.3 は，点 M_X, M_Y が回帰直線上にあるので座標系の原点を点 (M_X, M_Y) に移動することができ，その結果回帰直線の方程式が簡単になるという事実を表わしている．

あてはめ曲線として，直線ではなく**放物線**を使った場合には，近似曲線の形は次のようになる．

$$Y = a_0 + a_1 X + a_2 X^2 \tag{15}$$

この場合の正規方程式は次のようになる．

$$\begin{aligned}
\sum Y &= a_0 N + a_1 \sum X + a_2 \sum X^2 \\
\sum YX &= a_0 \sum X + a_1 \sum X^2 + a_2 \sum X^3 \\
\sum YX^2 &= a_0 \sum X^2 + a_1 \sum X^3 + a_2 \sum X^4
\end{aligned} \tag{16}$$

上記の原則は，どのような次数の多項式によって表わされる曲線に対しても拡張される．第 1 番目の正規方程式は常に，近似曲線の要素全体の総和になっている．第 2 番目の方程式は，近似曲線の各項と X との積の総和になっている．引続く各正規方程式は，近似方程式に引続く X のべき乗をかけ，積の総和をとれば求まる．上に示されたように，近似直線と放物線の場合にもこの法則が満足されることが簡単にわかる．

簡単な回帰直線についての一般化のもう一つの形は，（単に X, Y だけでなく）数個の変数がある時に“線型のあてはめ”を行なう場合である．3 変数 X, Y, Z の場合には，相関図は 3 次元になり，線形近似は**平面**となってその方程式は

$$Z = a_0 + a_1X + a_2Y \tag{17}$$

ここで a_0, a_1, a_2 は次の正規方程式を満足する.

$$\begin{aligned}\sum Z &= a_0N + a_1\sum X + a_2\sum Y \\ \sum ZX &= a_0\sum X + a_1\sum X^2 + a_2\sum YX \\ \sum ZY &= a_0\sum Y + a_1\sum XY + a_2\sum Y^2\end{aligned} \tag{18}$$

これらの式は，前述の，2変数を直線のかわりに放物線で近似する場合に似た法則に従うものとして記憶すればよいことに注意しよう.

簡単な2変数の線型あてはめを，高次の曲線や，多変数間の関係のどちらに拡張する場合でも，近似曲線や曲面の方程式における係数は，今までに見たように線型連立方程式を解くことによって得られる.

2-10　回帰直線と相関についての追記

データ Y とその平均値 M_Y との平均二乗誤差は，次の形で表わされる：

$$\sum (Y - M_Y)^2 = NS_Y{}^2 \tag{1}$$

左辺の総和は，(代数により)次の2つの総和で表わされることがわかる.

$$\sum (Y-M_Y)^2 = \sum (Y - Y_e)^2 + \sum (Y_e - M_Y)^2 \tag{2}$$

ここで，右辺の最初の総和はしばしば**非説明変分** (unexplained variation) と呼ばれ，2番目の総和は**説明変分**(explained variation)と呼ばれる．これらの呼称は以下の説明で理解されるであろう．データ Y に対して a_0, a_1 が見つかった後には，回帰直線，したがってその上のすべての点は完全に決定される．したがって，差分 $(Y_e - M_Y)$ は，直線上の値と平均値との間の差となる．しかしながら差分$(Y - Y_e)$は，実際のデータ Y と回帰直線との偏差を表わしており，明確な法則に従う必要はない．(それゆえ“非説明”である).

総変分に対する説明変分の比は

$$r^2 = \frac{\text{説明変分}}{\text{総変分}} = \frac{\sum (Y_e - M_Y)^2}{\sum (Y - M_Y)^2} \tag{3}$$

さて，この比が，2-8節で論じられ，その節の式(10)で与えられた相関係数(これも r と呼ばれた)の二乗に他ならないということを示そう．このことを示すために，上の式(3)から始めて，これを代数的に操作し，r^2 に対するモーメ

ント積の公式 [2-8 節, 式 (10)] に等しくしよう．第一に，式 (3) の分母は $\sum y^2$ であることに注意する．式 (3) の分子の総和における差の二乗は，2-9 節の式(12)を使って，変数 x, y で表わされる．式(3) はしたがって次のように書ける．

$$r^2 = \frac{\sum {a_1}^2 x^2}{\sum y^2} = \frac{{a_1}^2 \sum x^2}{\sum y^2} \tag{4}$$

2-9 節の式(13)によって ${a_1}^2$ を x, y の形で置きかえ，簡単化すれば

$$r^2 = \frac{\left(\sum xy\right)^2}{\left(\sum x^2\right)\left(\sum y^2\right)} \tag{5}$$

となり，これは 2-8 節の式(10)に等しい．また式(5) の x と y に関する対称性に注意しよう．このことは，Y に対する X の相関係数は X に対する Y のそれに等しいことを意味している．

式 (5) は，r^2 を表わす比較的簡単な代数的形式であるけれども，x と y を使って表わされており，X と Y の平均値を前もって計算しなければならないので，少なくとも 2 つのプログラム・ループが必要である．r^2 を表わす別の形によれば，単一のループ内で，r^2 をデータ・ベクトル X, Y から計算することができ，平均値と標準偏差もこのループで計算できる．

$$r = \frac{\left(N \sum XY\right) - \left(\sum X\right)\left(\sum Y\right)}{\sqrt{\left[\left(N \sum X^2\right) - \left(\sum X\right)^2\right]\left[\left(N \sum Y^2\right) - \left(\sum Y\right)^2\right]}} \tag{6}$$

次にあげるのは，相関係数の有益な性質である．

定理 2-10.1　推定の標準誤差 E, すなわちデータと回帰直線との差の二乗の平方平均は，相関係数と標準偏差だけを使って表わされる．

$$E^2 = (1 - r^2){S_Y}^2 \tag{7}$$

証　明

定義により

$$E^2 = \frac{1}{N} \sum (Y_e - Y)^2 = \frac{1}{N} \sum (y_e - y)^2 \tag{8}$$

ここに，$y_e = Y_e - M_Y$，$y = Y - M_Y$ である．また，2-9 節の式(12) より

$$y_e = a_1 x \tag{9a}$$

$$a_1 = \frac{\sum xy}{\sum x^2} \tag{9b}$$

式(8)の右辺の二乗を展開し，式(9)を代入すれば

$$E^2 = \frac{1}{N}\left[\frac{(\sum xy)^2}{\sum x^2} - \frac{2(\sum xy)^2}{\sum x^2} + \sum y^2\right] \tag{10}$$

同類項をまとめ，r についてのモーメント積の公式［式(5)］を使えば，式(10)は次のように書かれる．

$$E^2 = \frac{1}{N}(1 - r^2)\sum y^2 = (1 - r^2){S_Y}^2 \qquad \text{：証明終り} \tag{11}$$

式(7)から次のことがわかる：

1. $r = 1$ または-1 ならば $E^2 = 0$ である；すなわち，すべてのデータ点は，回帰直線上にある．
2. $r = 0$ ならば，推定の標準誤差は，Y の標準偏差に等しい．

r を**相関係数**と呼ぶことは，r はデータ X と Y との関連の度合を表わすある種の尺度であると解釈できることを示唆している．以下の諸点は，このことについて述べられたものである．これらを，上記の公式やそれから導出される式を使って確かめよ．

1. r の範囲は$-1 \leq r \leq 1$ である．
2. r が1に近いならば，X が増加するに従い，Y も線型に増加する傾向にあることを意味している．r が-1 に近いならば，X の増加に従い，Y は線型に減少する傾向にある．（どちらの場合にも，**X が Y の原因である，またはその逆であるという裏の意味があるわけではない**．しかし，それらに線型の因果関係があれば，r は$+1$ または-1 に近づくであろう．）
3. r が0に近いならば，X と Y のデータの間にはあまり**線型の**関係がないということを意味している．しかしながら，直線以外のある種の曲線による関係がやはりあるかも知れない．

演 習 問 題

1. 連続する5つのジョブ・デックによって読みこまれたカード枚数は次のようであったとする．

80, 19, 276, 58, 712

（a） デック・サイズの範囲を求めよ．

（b） デック・サイズの平均値を求めよ.

（c） 変動係数を求めよ.

（d） メディアンを求めよ.

2. ある計算機センタで処理されたジョブの1月，2月，3月それぞれについての平均実行時間だけが入手できるものと仮定する.

（a） このデータだけを使ってこの3ヵ月間の平均実行時間を計算できるか？

（b） もしできないならば，どんな情報を付け加える必要があるかを述べよ.

3. 数の3つの集合の平均値を求める公式を考えよ [2-1節の式 (9) に似ている].

4. データ X を受けとって以下の値を計算しプリントするプログラムを書け.

（a） 範囲

（b） 平均値

（c） 標準偏差

（d） 変動係数

（e） メディアン

5. データ X に対して度数によって分類された分布関数を作成するプログラムを書け. そのプログラムは，ユーザに対し彼の希望するバケットの数あるいはバケットの範囲のどちらかを選んで規定することを要求しなければならない. そのプログラムは次のものを計算しプリントしなければならない.

（a） 問題4のすべての値（したがって問題4のプログラムはこの問題のプログラムに含まれる).

（b） 以下の例のような形式の度数による分類表.

バケット番号	バケットの範囲	度　数	相対度数	累積度数	相対累積度数
1	0 ～0.5	10	0.0500	10	0.0500
2	0.5～1.0	50	0.2500	60	0.3000
⋮	⋮	⋮	⋮	⋮	⋮

6. 図 2-5.1 の一様乱数生成器(あるいは利用できる他の乱数生成器)を使って 1000 個の値を生成せよ.

（a） このデータを使って問題 5 で作ったプログラムを走らせよ. 結果の値をあなたが直観的に予想した値と比較してみよ.

（b） 最初の 500 個の乱数を X の値, 残りの 500 個を Y の値と考える. 相関係数を計算せよ.

7. いくつかのプログラムを観測して, 次のような主記憶装置の大きさ (k バイト)の相対度数関数が得られた.

k バイト	0	8	16	32	64
相対度数	0.00	0.80	0.10	0.06	0.04

（a） 相対累積度数関数を計算せよ.

（b） この関数を使って 1000 個のランダムな標本を生成せよ.

（c） バケットの大きさを適切(= 8)に設定して問題 5 で作ったプログラムを走らせ, 生成された数の相対度数関数を求めよ.

（d） 2 つの相対度数関数の差異の尺度として平均二乗差の平方根 (差の rms 値)を使って(c)で得た表と上で与えられた表とを比較せよ.

8. 計算機施設では数種類のデータを集めることができる (また, 集めなければならない). たとえば, さまざまなカテゴリ (コンパイル, ラン, ユーティリティ等)における実行時間, 実行待ち時間, ジョブの数などである. そのようなデータは, “システムが今何をしているか” をわれわれが知るのに役立つし, いかなる計算機センタの評価やその後の改善における非常に重要な第 1 ステップである.

この問題および問題 9, 10, 11 は非常に簡単な例――ジョブ実行時間の統計――を使ってジョブ・データ解析の考え方を示したものである. ある計算機センタで観測されたジョブの実行時間 (分) が 50 個, 次のように与えられている(ベクトル Y).

（a） このデータをソートしてベクトルで表わせ (目で見て調べるために).

（b） この実行時間データに対し, 前に作成したプログラムを使って問題 4 の 5 つの統計量を計算し, プリントせよ.

（c） 任意の都合のよいバケットの方式を使って, このデータの相対度

	10	18	91	42	13	11	32	17	4	152
	23	13	36	101	1	14	2	23	34	15
$Y =$	27	1	57	17	3	29	53	4	62	47
	4	11	20	13	38	54	46	12	5	26
	18	19	3	10	5	13	24	3	2	11

数表と相対累積度数表を計算して印刷せよ.

(d) ヒストグラムを描け.

9. 問題8の計算(および他の計算)から，以下の問に答えよ．ただし，ジョブは実行時間が非減少の順に並べてあると考える.

(a) 実行されたジョブ総数の20%，40%，60%，80%のうちでそれぞれ最短のジョブのなかで最長のものはどれか？

(b) すべてのジョブを実行するのに必要な**総時間**の20%, 40%, 60%, 80%を占めるジョブのうちでそれぞれ最短のもののなかから，最長のものはどれか？

(c) 総実行時間の20%，40%，60%，80% を占めるジョブの個数はジョブ総数のそれぞれ何パーセントにあたるか？

10. 問題8に実行時間が与えられているジョブはそれぞれ次に掲げる大きさ(k バイト)の主記憶装置を要求する(ベクトル S).

	80	120	250	200	90	50	60	100	20	160
	100	90	60	64	18	160	36	80	180	76
$S =$	90	24	48	120	44	170	220	20	260	190
	100	76	120	80	128	200	190	128	100	64
	100	120	64	80	16	90	80	68	36	76

(a) Y に対する S の相関図を描け．実行時間と記憶装置空間のデータを解析するための次のようなパラメータを計算し印刷するプログラムを書け.

(b) 最良の最小二乗直線の a_0, a_1 を決定し，相関図上に直線を描け.

(c) 推定の標準誤差はいくらか？

(d) 相関係数はいくらか？

(e) 与えられたデータで表わされたこのジョブ・ストリームの時間と空間との要求の間にはどのような関係があるとあなたは結論するか？

11. 問題8から10までの結果をレポート(1, 2枚)にまとめよ.

12. カード・コンパクションの問題: 50人位の学生のジョブや他のジョブを集めて, 穿孔カード(あるいは磁気テープ上のカード・イメージ)のいくつかのデックを得る.

(a) カード上の第1欄から最後のブランク以外の文字までの文字数で定義される各カードの使用長さを測定するプログラムを書け.

(b) デックに対して度数関数と累積度数関数を作れ. また, 相対度数と相対累積度数を計算せよ.

(c) カードの使用長さの平均値はいくらか？

(d) 次のような文字を詰めるプログラムを考えよ. すなわち, 各カードをあるバッファ・エリアに読み込み, 使用長さを求め, それを1バイトに格納し, その後に最後のブランク以外の文字までのカード文字を格納する. 各カードの80個の欄をすべて格納するのにくらべ, 格納する文字数がどれ位節約されるかを見積れ. また, 節約率を計算せよ.

参 考 文 献

1. [DA] Daniel, C. and F.S. Wood: "Fitting Equations to Data," John Wiley and Sons, Inc., New York, 1971.
2. [DR] Draper, N.R. and H. Smith: "Applied Regression Analysis," John Wiley and Sons, Inc., New York, 1967.
3. [FR] Freiberger, W. (Ed): "Statistical Computer-Performance Evaluation," Academic Press, New York, 1972.
4. [HA] Hahn, G.J. and S.S. Shapiro: "Statistical Models in Engineering," John Wiley and Sons, Inc., New York, 1968.
5. [HAN] Hanssmann, F., W. Kistler, and H. Schultz: Modelling for Computing Center Planning, *IBM Systems Journal*, vol. 10, no. 4, 1971.
6. [HI] Hickman, E.P. and J.G. Hilton: "Probability and Statistical Analysis," Intext Publ. Co., 1971.
7. [IB] IBM Corp.: "Random Number Generation and Testing," Document #C20-8011, IBM Corp., 112 E. Post Rd., White Plains, N.Y.
8. [KN] Knuth, D.E.: "The Art of Computer Programming," vol. 2, Seminumerical Algorithms, Chapter 3, Random Numbers, Addison-Wesley Publishing Company, Reading, Mass., 1969.
9. [MA] Margolin, B.H., R.P. Parmalee, and M. Schatzoff: Analysis of Free-Storage Algorithms, *IBM Systems Journal*, vol. 10, no. 4, 1971.

10. [PA] Parzen, E.: "Modern Probability Theory and Its Applications," John Wiley and Sons, Inc., New York, 1960.
11. [SP] Spiegel, M.R.: "Statistics," Schaum's Outline Series, McGraw-Hill Book Company, New York, 1961.
12. [ST] Stanley, W.I. and H.F. Hertel: Statistics Gathering and Simulation for the Apollo Real-Time Operating System, *IBM Systems Journal*, vol. 7, no. 2, pp. 85-102, 1968.

第3章

離散数学とその応用における話題

ディジタル計算機は本質的には，いろいろな型の“系列”に並べられた離散記号の処理機械である．たとえば，数の表現は，ディジット，符号，および句切り記号の系列である；1つのプログラム全体でさえも，記号の長い系列であると考えられる．離散数学は，記号系列のいろいろな理論をあつかう数学の分野であり，おのおのの記号は，あらかじめ規定されたあるアルファベットの要素である．計算機の分野では，このアルファベットを**文字セット**と呼ぶ．離散数学の話題は，順列，組合せ，再帰，そして集合と確率のいくつかの(全部ではない)理論を含んでいる．定理の証明もまた記号の系列であり，それゆえ証明の理論も離散数学の話題であるということは注目すべきことでもある．

この章では，離散数学における話題を選んで，短いが厳密な記述を与えている．それにはまた，微積分学のような，連続数学の技法を使う場合もあるだろう．われわれの目的は，次のものである．

1. 計算機システムの性能を含めて，システム解析の理論の多くの基礎をなしている考え方を説明すること．
2. ある種の表記法や結果を導き出し，列挙すること．これらのいくつかは，この本で後に使われる．他のものも，この本で使われてはいないが，一般的な興味があり，読者が文献の中の理論に関する論文を勉強する時に助けとなるであろう．

最初の数節では，組合せ論，すなわちある規則に従って作られた記号系列を数えあげる方法，の原理について論じる．次に，集合論に話を進め，そこから関数と再帰について考える．組合せ論と集合論はともに，それら自身がもともと重要な学問であるというだけでなく，この章の主要な話題である確率論に欠

くことができないものでもある．この章のいたるところで，計算機システムはもちろん日常生活からも，例をとりあげるであろう．この章の最後の節では，システム解析への確率の応用の一つである，信頼性の理論に焦点をあてる．

3-1 組合せ論の基礎：順列と組合せ

これ以後，全体を通じて，系列を形成するのに使われるすべての可能な記号を並べあげたアルファベットが，前もって決められているものと仮定する．アルファベットの**サイズ**ということばは，単にそれが含んでいる記号の個数を意味する．たとえば，記号系列が，1個のサイコロを投げた時に出た目の記録である場合には，アルファベットは {1, 2, 3, 4, 5, 6} であり，アルファベットのサイズは6である．

記号系列を形成するためのある規則が決められているものと仮定しよう．この規則を使って，相異なる系列がいくつ形成できるだろうか．**相異なる**という意味は，どんな規則が与えられていても，同一の系列を2回以上は数えないということである．他に何も言われていなければ，**系列の個数**という言葉は，**相異なる系列の個数**という意味である．系列を形成する規則は無数に存在しうるが，非常によく出てくるので，強調しておく価値のある2つの性質がある．

1. **記号の重複**：1つの系列内で，同一の記号が2回以上表われることが許されている(許されていない)．
2. **順序の重要性**：同じ記号が異なる順序に並んだものは，数えあげるべき相異なる系列と見なされる(見なされない)．

これらの性質は重要で有用であるが，組合せ問題を解く万能薬というものはないことを，読者に注意しておく．この問題には，落し穴がいっぱいある；問題の文章が見かけ上ちょっと変わっただけで，答が著しく変わるばかりでなく，異なった解法が必要となることもある．

基礎となる考えは，**積の規則**である．なぜなら後に出てくる公式のいくつかは，注意深くこの規則を使って導出されるからである．

定理 3-1.1 積の規則　それぞれ n_1, n_2 個の記号を持つ2つのアルファベット A, B が与えられている．アルファベット A からとった1番目の記号と B からとった2番目の記号とから形成されうる長さ2の系列の個数は，

$$N = n_1 n_2 \tag{1}$$

系列(常に，相異なる系列を意味する)の数についてのこの公式は，系列を生成する過程を想像すれば理解できるだろう．最初の記号は n_1 個のうちのどれかから選ばれる．これらの**おのおのに**対し，2番目の記号は n_2 個のうちのどれかから選ばれる．したがって，長さ2の可能な系列は，$n_1 \times n_2$ 個ある．

例：　硬貨を投げてからサイコロをふった時，それらの面の組合せは，何通り可能だろうか．

解：　硬貨については，アルファベットは $\{h, t\}$ である．サイコロについては，$\{1, 2, 3, 4, 5, 6\}$ である．したがって $n_1 = 2$，$n_2 = 6$ である．式(1)により，

$$N = 2 \times 6 = 12$$

これら12個の系列を数えあげると

$$\{h1, h2, h3, h4, h5, h6, t1, t2, t3, t4, t5, t6\}$$

このモデルと上記の結果は，各試行が，“硬貨を投げてからサイコロをふる”という順序で行なわれることを必要としていることに注意しよう．$1h$ のような系列が決して現われないのは，このためである．

定理3-1.1の自然な拡張は，次の定理である：

定理3-1.2　サイズが $n_1, n_2, \cdots, n_k$ の k 個のアルファベットが与えられている．引続くアルファベットから1つずつ記号を選んで，一定の順序で並べて作った長さ k の系列の個数は

$$N = \prod_{i=1}^{k} n_i \tag{2}$$

この結果の非常に重要な特例は，すべてのアルファベットが同じで，それゆえ n_i がすべて等しい場合である．その場合には，同じアルファベットを一貫して使う．

定理3-1.3　n 個の記号のアルファベットから，**重複を許して**作られた長さ k の系列の個数は

$$N = n^k \tag{3}$$

この場合の重要性は，次の例で示される．10個の10進数記号から成るアル

ファベット($n=10$)を考えよう．

$$D=\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

たとえば k が 3 ならば，式(3)は，10^3 個の可能な系列があることを示している．それらは：000, 001, $\cdots$, 999，すなわち，可能なすべての 3 桁の 10 進数である．したがって，定理 **3-1.3** は，われわれが用いている位取り整数系で暗黙のうちに使われている数え上げの方法の核心であることがわかる．別の例として，次のような本を考えよう．

ページ数 $=500$
1 ページあたりの行数 $=44$
1 行あたりの文字数 $=72$
アルファベットのサイズ $=70$

ここでは，アルファベットは，大文字，小文字，数字，空白，いろいろな区切り記号を含んでいる．このような本はどれでも，長さが

$$k=500\times 44\times 72=1584000$$

の長い記号系列であると考えることができる．したがって，このクラスの可能な本の総数は

$$N=70^{1584000}$$

この数の大きさは，人間の理解を超えている．それは，太陽系にある電子の総数よりもずっと大きいのである！しかし，それは依然として有限な数である．さらにわれわれは，そのような系列のすべて，したがってそのような本のすべてを，組織的に数え上げる方法（もっとも，そうするには，悠久の時間が必要となるが）を知っている．これらの本は，長さが k でアルファベットのサイズが 70 であるような，今までに書かれたか，あるいは将来にわたって書かれるであろうすべての本を含んでいる！

定理 **3-1.3** は，記号の重複が許されている系列を数えあげる；それが含む各記号の個数が同じでも順序が異なる系列は，すべて相異なる系列として数えあげられる．われわれは次に，重複がやはり許されているが，その並べかえを勘定に入れない場合を考えよう．

定理 3-1.4　n 個の記号のアルファベットから重複を許して作られる長さ k の系列の個数は，同じ記号が異なる順序で並んだものを別の系列として数え**な**

い場合には次のようになる．

$$D(n, k)=\frac{(n+k-1)!}{k!\,(n-1)!} \tag{4}$$

この数はしばしば，n **個のものから，重複を許して** k **個をとり出して作る組合せの数**と呼ばれる．この場合には，2つの系列 A, B の間で，少くとも1つの記号の出現回数が異なっている場合にのみ，これらの系列は別のものとして数えられる．たとえば，次の2つの系列を別のものとして数えることは**ない．**

$$abcbc$$
$$accbb$$

次に，1つの系列で同じ記号の**重複が許されていない**ような系列を作ることを考えよう．

定理 3-1.5　順列　　サイズが n のアルファベットから作られる長さ k の系列の個数は，系列内での**重複を許さず**，また，順序が異なるものは別の系列として数えられる場合には次のようになる：

$$P(n, k)=n(n-1)(n-2)\cdots(n-k+1) \qquad 0\le k\le n \tag{5}$$

この結果は，式(1)の積の規則を使えば，簡単に理解できる．系列の作成を始めるには，アルファベットのうちの n **個の**どれでも，1番目の記号として選ぶことができる．そのような選択のどれに対しても，重複が許されていないので，2番目の記号を選ぶ方法は，$n-1$ 通りしかない．積の規則によって，初めの2つの記号を選択できる方法は，$n(n-1)$通りとなる．後続する記号に同じ考えを適用すれば，式(5)が得られる．

式(5)の$P(n, k)$は，しばしば，n **個のものから，（重複なしで）**k **個を取り出して作る順列の個数**と呼ばれる．この公式は，しばしば階乗関数によって表わされる．

$$P(n, k)=\frac{n!}{(n-k)!} \qquad 0\le k\le n \tag{6}$$

式(6)の再帰形は簡単に導かれる：

$$P(n,0)=1$$
$$P(n, k)=(n-k+1)P(n, k-1), \qquad 1\le k\le n$$

例：　ある計算機設備において，ある1時間に20個のジョブが走る．

これら20個のジョブから一定の順序でとられた**5**個のジョブから成るジョブ・ストリームのサンプルを選ぶ．このようなサンプル・ストリームはいくつあるか？

解：　ジョブには，1, 2, ⋯, 20というラベルが付けられていると考える．したがってこれがわれわれのアルファベットである．長さ**5**の系列の個数は，式(6)から

$$P(20,\ 5) = \frac{20\,!}{(20-5)\,!} = \frac{20\,!}{15\,!} = 20\cdot19\cdot18\cdot17\cdot16 = 1860480$$

式(6)の重要な特例は，**相異なる n 個のものを（重複なしで）並べる方法が何通り可能であるか**を知りたい場合である．式(6)で $k=n$ とすれば

$$P(n,\ n) = \frac{n\,!}{0\,!} = \frac{n\,!}{1} = n\,! \tag{7}$$

順列を作る方法は，トリーとして思い浮かべられるだろう(**図 3-1.1**)．各系列は，トリーのノード・ラベルを左から右へと読むことにより，数えあげられる．

順列のもう**1**つの変形は，先頭に来るべき特別な記号を持たない円順列の個数を知りたい場合である．長さ n のそのような系列の個数は，$(n-1)\,!$ となる．

順列として数えられたどの系列においても，同じ記号は**2**回以上出現しない．同じ記号が異なる順序に並んだものは，別の系列として数えられる．後者の性質は，**組合せ**を求める時には省かれる．

定理 3-1.6　組合せ　サイズが n のアルファベットから作られる長さ k の系列の個数は，系列内での**重複を許さず**，また，他のある系列内に現われるすべての記号を持つ系列がない（すなわち同じ記号が異なる順序で並んだものは**許されない**）場合には，次の式で求められる：

$$C(n,\ k) = \frac{n!}{k!(n-k)!} \qquad 0 \le k \le n \tag{8}$$

また

$$C(n,\ k) = \frac{P(n,\ k)}{k!} \tag{9}$$

組合せの再帰形は

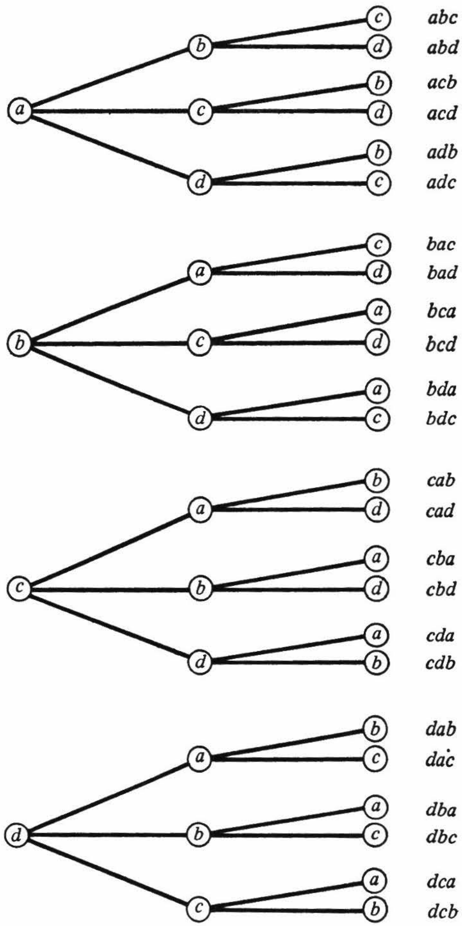


図 3-1.1　順列トリー　n = 4,　k = 3

$$C(n, 0) = 1 \tag{10a}$$

$$C(n, k) = \frac{n-(k-1)}{k} C(n, k-1), \qquad 1 \leq k \leq n \tag{10b}$$

文献で時々見られる，$C(n, k)$の他のいくつかの表記法は，

$$C(n, k) = \binom{n}{k} = nCk = \{k\}_n$$

$C(n, k)$は，組合せ論で広く使われる基本的な関数である．それは，驚くほど数多くのいろいろな文章に出てくるが，ここでは，それらのうちの少しだけを議論しよう．

式(8)がどのように導出されるかを理解するには，定理3-1.5から，順序も考慮に入れて数えた，長さkの系列(重複は許さない)の個数は，$P(n, k)$であることを思い出そう．さて，順列として列挙されたもののうちのどれか1つの系列を考えよう．この系列のk個の相異なる記号は，サイズがkのアルファベットと考えられ，したがって式(7)から，$k!$通りの方法で並べかえることができる．それゆえ同じ記号から成るおのおのの系列は，同じ記号が異なる順序で並んでいる$k!$個の系列から成る**集合**に属している．そのような**集合**の個数は，$P(n, k)$を$k!$で割れば求まり，したがって式(9)となる．

例：　5つの整理だなと3つの物が与えられている．1つの物は，1つのたなに置かれるものとする．これを行なうには，何通りの方法が可能か？

解：　整理だなに，1, 2, 3, 4, 5とラベルを付ける．1つのたなにつき物が1つ以下なので，ある1つの“方法”は，5個の記号のラベル・アルファベットから重複なしに選ばれた3つのラベルから成る1つの系列に対応する．したがって，式(9)により

$$C(5,3) = \frac{5!}{3!\,2!} = 10$$

この問題のもう1つの考え方は，まず，整理だなの占有状態のすべてを列挙することである．これを行なうには，整理だなのラベル(上の例では，1, 2, 3, 4, 5)が列の見出しになっている一種の真理値表(図3-1.2)を書けばよい．この表の0, 1から成る1つの行が，おのおのの占有状態であり，1はそのたなが占有されていることを示している．1がちょうど3個ある状態の集合が，ちょうど3個のたなが占有されているすべての状態を表わしている．したがって，

$C(n, k) =$ ちょうどk個の1，$(n-k)$個の0から成る，
長さがnビットの可能なビット・ベクトルの個数

与えられたnとすべてのkに対して列挙した，完全な真理値表を書けば，次の重要な公式が正しいことがわかる．

$$\sum_{k=0}^{n} C(n, k) = 2^n \tag{11}$$

占有状態：五つの整理だなのうちの三つが占有される

1	*2*	*3*	*4*	*5*	← 整理だな
0	0	1	1	1	
0	1	0	1	1	
0	1	1	0	1	
0	1	1	1	0	
1	0	0	1	1	
1	0	1	1	0	占有状態
1	0	1	0	1	
1	1	0	0	1	
1	1	0	1	0	
1	1	1	0	0	

$n=5, k=3$の場合の占有状態の数
$c(5,3)=10$

図 3-1.2　3つの物による5つの整理棚の占有状感

やはり重要な$C(n, k)$のもう1つの意味は，それが，二項式$(x+y)^n$の展開におけるk番目の項の係数（一番左端の項を$k=0$とする）になるということである．

$$(x+y)^n = \sum_{k=0}^{n} C(n, k)x^{n-k}y^k \tag{12}$$

例：
$$(x+y)^4 = 1x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4$$

$C(n, k)$がしばしば**二項係数**(binomial coefficient)と呼ばれるのは，$C(n, k)$に上記の性質があるからである．

フランスの数学者 Blaise Pascal(1623-1662)は，$C(n, k)$の値を三角形（ピラミッド）で表わすというおもしろい方法を提案した．第n行（頂点で$n=0$と数える）は，$k=0, 1, \cdots, n$の$n+1$個のkの値に対するすべての$C(n, k)$の値を与えている．きちんと書けば，三角形の各項（端にあるものを除く）は，その上の行にある隣接する2つの項の和になっている（**図 3-1.3** 参照）．

n個のものから一度にk個をとり出して作る組合せの個数の，1つの一般化が次に述べる定理に含まれている．

定理 3-1.7　多項式(multinomial)　n個の記号のアルファベットから長さkの系列を作るものとし，この場合に，各系列が第1番目の記号はちょうどk_1

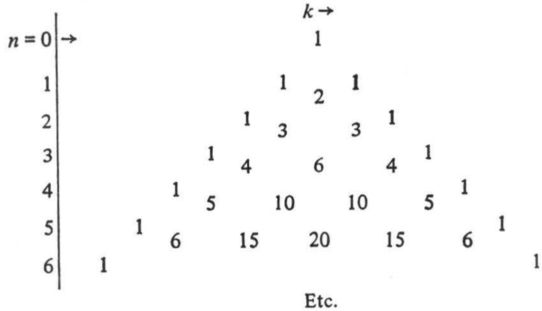


図 3-1.3 $C(n, k)$に対する Pascal の三角形

回，2番目は k_2 回, $\cdots$, と，重複して含んでいなければならないとする．相異なるこのような系列の個数は

$$M(k_1, k_2, \cdots, k_n) = \frac{k!}{k_1!\, k_2! \cdots k_n!} \tag{13 a}$$

ここに

$$k = \sum_{i=1}^{n} k_i \tag{13 b}$$

多項式の特別な場合は，すべての k_i が，たとえば m に等しい場合である．この場合，式(13 a)は次のようになる：

$$N(m, n) = \frac{(nm)!}{(m!)^n} \tag{14}$$

例: 20人の選手によるブリッジのトーナメント試合を，4人ずつの5つのグループに分けて行なうことにする．何通りの方法でこれを行なうことができるか．

解: 各グループに，A から E までの文字を割りあてたとしよう．ある1つの“方法”は，$nm = 20$ 個の記号(1人につき1個の記号)から成る1つの系列に対応し，それぞれの系列は，$n = 5$ のアルファベットから各記号を $m = 4$ 回重複して選んだものとなっている．このような系列の個数は，式(14)によって

$$N(4, 5) = \frac{(20)!}{(4!)^5}$$

しかし，これは完全な答ではない．というのは，単にラベルが異なるだけでメンバーは同じであるグループに人を割りあてる系列を，別の系列として数えているからである．**5**つのグループにラベルを付ける方法は**5!**通りあるので，答は

$$\frac{(20)!}{(5!)(4!)^5}$$

3-2　集合：定義，規定，同等，同値［ST 参照］

集合論は，物の集まりの正確な性質を取扱う．集合は，数学体系の階層構造において基本的な位置を占めているので，しばしば，形式的，公理的な方法で紹介される．われわれはこのような扱い方をわずかながら反映するが，主に，その上である演算子が定義されている基本的なデータ構造として，集合を取り扱う．

定義　限定された識別可能な，われわれの直観または思考の対象を1つのまとまりと考えて集めたものを集合 S という．その対象は集合 S の**要素**または**元**と呼ばれる．

この定義は，集合の元としてどんなものが許されるかについて，かなり自由に考えてもよいことを示している．われわれの例のほとんどは，集合を，**記号の集まり**と考えている．他の論文では，集合は，直線上の点，幾何図形，人間等の集まりであるかもしれない．**限定された**ということばは，対象 x と集合 S が与えられれば，いつでも x が S の元であるかどうかを決定することができる，ということを意味している．この考えは，次のような演算子$\in$を定義すればもっとはっきりする．すなわち文 $x\in S$ は，x が本当に S の元であれば真，そうでなければ偽の値をとる．この演算子は集合論において基本的なものである；すなわち，この演算子は，より基本的な演算子によって定義することができない．しかし，文 $x\in S$ は，もう少し違った意味でよく使われており，われわれはこちらを採用する：すなわち，$x\in S$ は，x が“S の元である”ということを言明しており，また，$x\notin S$ は，x が“S の元でない”ことを表わしている．

集合を規定するいくつかの基礎的な方法は次のものである：

1. 元を列挙したリストを作る．

例：　$A = \{a, b, 1, c, d\}$

元は，通常小文字や数字で書かれ，集合の名前は大文字で与えられる．集合の元は，リストの中ではコンマで区切られ，リストは中かっこでくくられる．

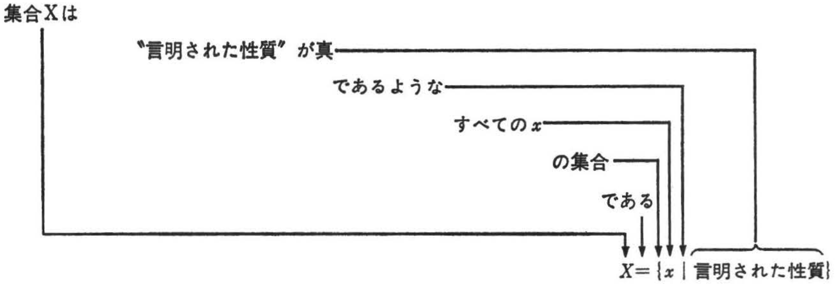
2．集合のすべての元が満足する性質を述べる．

例：　$A =$ {その地方の中学校の最高学年にいる少年の全部}

この型の規定は，元の数が大きい時にはリストを作るよりずっと便利であり，元の数が無限である時には，欠くことができない．このように，われわれは

$$Z = \{\text{すべての整数}\}$$

とは言うであろうが，Z のすべての元のリストを作ることはもちろんできない．
性質を述べる型の規定は，通常次の表記方法を使って作られる．



例：　$X = \{x|x > 5\}$
意味：X は，"x は5より大きい"ということが真であるようなすべての x の集合である．すなわち，X は5より大きいすべての x の集合である．

3．集合は，すでに存在している集合に，ある演算子のどれかを適用して得られたものとして規定されてもよい．（これらの演算子については，後で簡単に議論する．）

4．2つの特別な集合には，次の独特な名前が与えられている：

ϕ　**空集合**(empty or null set)　　元を持たない集合

U	**普遍集合**(universal set)	与えられた理論におけるすべての集合のすべての元を含む集合

2つの集合を**同等**(equal)である$(A = B)$と定義するための必要十分条件は，それらが同じ元を持つことである．$A = B$ならば，Aのすべての元はBの元であり，Bのすべての元はAの元である．このことによって，集合の元を並べた順序には意味が**ない**．また，いくつかの元を重複して並べた集合は，同じ元が1回ずつだけ現われる集合と同等である．今後は，興味のあるすべての集合は，重複した元を持たないものと考えよう．今議論した特性は，次のように説明される：

$$\{a, b, c\} = \{a, c, b\} = \{a, c, c, b, a\}$$

ある集合は，他の集合の1つの元であってもよい．この結果，単純だが捉えにくい次の結論が生じる．

$$\{a, b, c\} \neq \{a, \{b, c\}\}$$

この理由は，左辺の集合は3つの元，a, b, cを持つのに対し，右辺の集合は2つの元，aおよび集合$\{b, c\}$，しか持たないし，さらにcは左辺の集合の元であるが右辺の集合の元ではないからである．別の例は

$$\{\{a\}\} \neq \{a\}$$

であり，この理由は，左辺の集合は，集合$\{a\}$という1つの元を持つのに対し，右辺の集合は，aという1つの元を持つからである．

2つの集合があり，一方の集合の元のおのおのを，他方の集合の1つの元と対にする，すなわち関連させることができるならば，すなわちどちらの集合にも“とり残される”元がないという意味で1対1の対応がつけられるならば，この2つの集合は**同値である**(equivalent)と定義される．同等と同値の定義を説明するために，次のような集合を考えよう：

$$A = \{a, b, c\} \quad B = \{c, b, a\} \quad C = \{p, r, q\} \quad D = \{a, b, c, q\}$$

このとき，集合AとBは同等であり，集合A, B, Cは同値であり，集合Dは，他の集合と同等でも，同値でもない．

同値は，**濃度**(cardinal number)を定義するのに直接使われる．この性質により，お互いに同値である集合は同じ濃度を持つと定義される．したがって，“3台の計算機，3人の学生，3人の政治家”という3つの集合は，集合論の

意味で同値であり，それらの濃度は3である．おのおのの濃度はそれゆえ整数に対応しており，各整数は同値な集合の1つのクラスの名前と考えられる．空集合の濃度は0である．

3-3　集合の演算子と式

集合についての，簡単だが非常によく知られた3つの演算子をこれから定義しよう．その後で，これらがどのように結合されて式を作り，したがって集合を組み合わせて新しい集合を作るための非常に強力で正確な言語を用意してくれるかということを示そう．

定義は，以前に述べた表記法を使って簡潔に述べられる．

和集合(union):　$A \cup B = \{x|x \in A \text{ or } x \in B\}$

共通部分(intersection):　$A \cap B = \{x|x \in A \text{ and } x \in B\}$

相対補集合(relative complement):　$A \sim B = \{x|x \in A \text{ and } x \notin B\}$

絶対補集合(absolute complement):　$\sim B = U \sim B$ (Uは普遍集合)

これらをことばで述べれば次のようになる：2つの集合A, Bの和集合は，AあるいはB(または両方)に含まれるすべての元の集合である：A, Bの共通部分は，AとB両方に共通なすべての元の集合である：Aに関するBの補集合は，Bに含まれないAのすべての元の集合である：しばしば見られるように，Aが普遍集合である場合には，Aは表立っては記述されず，Bの絶対補集合と言われる．

集合演算子を視覚的に表わすためのよく知られている図式が，**Venn**図(図3-3.1)である．集合は(円などの)簡単な幾何図形で表わされ，演算子や式で述

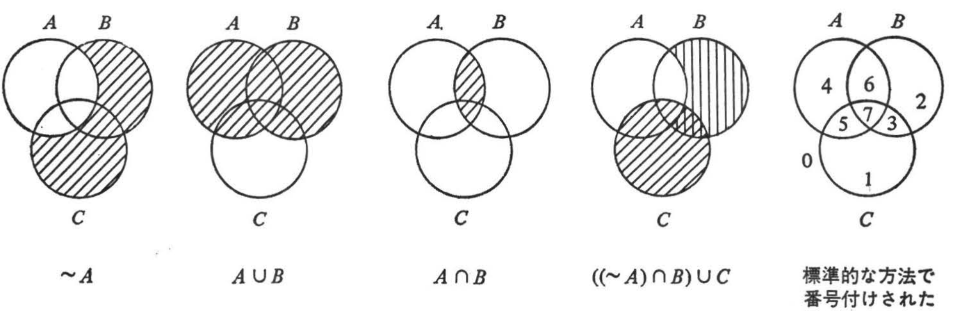


図 3-3.1　集合演算と標準的な番号付けを示す Venn 図

				注　釈
(a)	$A \cup \phi = A$	(a′)	$A \cap \phi = \phi$	ϕ は和集合に対する恒等集合である.
(b)	$A \cup \mathrm{U} = \mathrm{U}$	(b′)	$A \cap \mathrm{U} = A$	U は積集合に対する恒等集合である.
(c)	$A \cup A = A$	(c′)	$A \cap A = A$	和集合と積集合のベキ等法則
(d)	$A \cup \sim A = \mathrm{U}$	(d′)	$A \cap \sim A = \phi$	
(e)	$A \cup (A \cap B) = A$	(e′)	$A \cap (A \cup B) = A$	吸収法則
(f)	$\sim\sim A = A$			~の演算を2回施すと元の集合に戻る.
(g)	$A \cup B = B \cup A$	(g′)	$A \cap B = B \cap A$	和集合と積集合に関する可換法則
(h)	$A \cup (B \cup C) = (A \cup B) \cup C$			和集合と積集合に関する結合法則
(h′)	$A \cap (B \cap C) = (A \cap B) \cap C$			
(i)	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$			分配法則：積集合との和
(i′)	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$			和集合との積
(j)	$\sim(A \cup B) = \sim A \cap \sim B$			DeMorgan の法則
(j′)	$\sim(A \cap B) = \sim A \cup \sim B$			

図 3-3.2　集合演算子のいくつかの性質

べられたものの結果は，影をつけた部分で示される．与えられた集合のすべての部分集合を識別しておくことが便利である；集合が3つある場合に部分集合にラベルをつける方法が図に示されており，整数のラベルは前に述べた，ビット・ベクトルを使う部分集合のラベル付け方法に対応している．

包含関係を述べた“A は B の部分集合である”($A \subset B$ と書かれる)という言明が，$A \cup B = B$，または $A \cap B = A$，または $A \cap \sim B = \phi$，または $B \cup \sim A = U$ と同値であることを示すことができる．基本的な集合演算子は，集合の代数を定義するのに用いられる．この目的にとって，交換法則，結合法則，分配法則は，**図 3-3.2** に示されている他の性質と同様に非常に重要である．

集合，集合名，集合演算子は，普通の代数で行なわれるのと全く同じように結合されて式を形成する．次の順位規則によって，こみ入った式を書くのが簡単になる：

1. 補集合を作る演算が最初に行なわれる．

2. 共通部分を作る演算がその次に行なわれる.
3. 和集合を作る演算がその次に行なわれる.
4. かっこは，式の中の部分式を区切るのに使われる．上の規則は，それぞれの部分式の中に適用され，また部分式の結果を全体の式の残りの部分で使う時にも適用される.
5. 式の走査は，左から右へと行なわれる.

集合名は1文字とするならば(あるいは特にことわらなければ)，共通部分をとる演算子を省略し，$AB = A \cap B$ と書いてもよい．これを説明するために，左辺にこの規則を使った式を，右辺にはかっこを完全に使った等しい式を示す.

$$A \sim B(C \cup A \sim C) = (A \cap (\sim B)) \cap (C \cup (A \cap (\sim C)))$$

演算子は与えられたものから新しい集合を作り出すのであるから，結果の集合の元の個数が被演算集合の元の個数とどんな関係にあるかを見ることは興味深い．集合 A の元の個数を $N(A)$ で表わす．上で定義した3つの演算子について次のことが言える：

$$N(\sim A) = N(\mathrm{U}) - N(A) \tag{1}$$

$$N(A \cap B) \leq \mathrm{Min}(N(A), N(B)) \tag{2}$$

$$N(A \cup B) = N(A) + N(B) - N(A \cap B) \tag{3}$$

補集合の演算子の場合の式(1)は，補集合の定義から直ちに導かれる．共通部分の場合には精密な関係ではなく，その限度について一般的にしか言うことができない：結果の集合の元の個数はたかだか，元の2つの集合の元の個数の小さい方の大きさである．2つの集合の和集合の元の個数は，それぞれの元の個数から，2つの集合の共通部分の元の個数を減じたものである．この規則の論拠は，和集合についてのベン図(図3-3.1)から明らかである．この図は，集合 A, B の元の個数を加えれば，共通部分の個数が2回数えられてしまうことを示しており，したがって上の式に見られるように，2回数えられた元の個数を減じておかなければならない．共通部分が空集合であるような2つの集合は**互いに素**(disjoint)であると言われる．$N(A \cup B)$ についての式を見れば，互いに素な集合の和集合の場合にのみ，結果の集合の元の個数は，2つの被演算集合の元の個数の和に等しいことがわかる.

3-4 部 分 集 合

1つの集合 A のすべての元がもう1つの集合 B の元でもある場合に，A は B に含まれると言う($A \subseteq B$ と書く)．$A \subseteq B$ を“A は B の部分集合である”と読んでも同じ意味である．空集合 ϕ はすべての集合に含まれており，すべての集合は普遍集合に含まれている．A が B の**真部分集合**(proper-subset)である($A \subset B$ と書く)と言うための必要十分条件は，$A \subseteq B$ かつ $A \neq B$ であること，すなわち B は，A のすべての元およびそれ以外の元を少なくとも1つ含むことである．したがって，A のすべての元は A の1つの部分集合を成すが，それは真部分集合とはならない．

リストで表わされた集合が与えられた時に，そのすべての部分集合を系統的に列挙することは，次のような表を用いて簡単に行なえる．表の各列のラベルは，集合の元となっており，各行は，1つの部分集合を1, 0の系列で表わしている．1は，列の見出しの元が，部分集合に含まれていることを示し，0はそうでないことを示している．たとえば，集合が $\{a, b, c, d\}$ であれば，その部分集合 $\{a, c, d\}$ は1011という行で表わされる．与えられた集合内の n 個の元に対しては，行が整数の2進表現になっていることに気づけば，簡単にその表が作られる．これは，元が n 個の集合の部分集合の個数は 2^n であるということを理解する1つの方法である．ある与えられた集合 A の 2^n 個のすべての部分集合から成る集合は，A の**べキ集合**(power set)と呼ばれ，$P(A)$ と書かれる．

例:
$$A = \{a, b, c\}$$
$$P(A) = \{\phi, \{c\}, \{b\}, \{b, c\}, \{a\}, \{a, c\}, \{a, b\}, \{a, b, c\}\}$$

部分集合を系統的に列挙する方法は，多くの問題に役に立つ手法である．そのような問題の代表例として，次の簡単な問題を考えよう．

例:　計算機学科のある大学院のクラスには，100人の学生がおり，そのうち40人は工学専攻学生で，残りは教養学専攻学生である．25人の学生が優等卒業学位をもらったが，そのうち10人は工学専攻学生ではなかった．次のものを求めよ．

(a)　優等卒業学位をもらわなかった教養学専攻学生の数

(b)　優等卒業学位をもらった工学専攻学生の数

解:　まず適切な分類を明らかにすることからはじめる．その分類は学生の集合を規定しているのであるが，この問題に対する集合の元となる．そこで

E は工学専攻学生を表わし，H は優等卒業学位をもらった学生を表わす．次の表に，可能な部分集合が示され，それぞれには識別番号が割りあてられている．

部分集合の番号	E	H
0	0	0
1	0	1
2	1	0
3	1	1

言葉でいえば，部分集合 2 は“工学専攻であり，**しかも**優等卒業学位をもらわなかった学生の集合”である．他の部分集合も同じように言うことができる．部分集合 i の学生数は n_i で表わされる．問題の記述は，次の方程式で表わすことができる．

$n_0+n_1+n_2+n_3=100$ （学生は 100 人）
$n_2+n_3=40$ （工学専攻学生は 40 人）
$n_1+n_3=25$ （優等卒業学位をもらった学生は 25 人）
$n_1=10$ （優等卒業学位をもらった教養学専攻学生は 10 人）

この方程式を解いて $n_0=50$，$n_3=15$ が求めるものになる．

上の手法は，計算機システムのリソースをオーバラップして使用する場合の解析に貴重なものである．4-6 節にその例がある．

共通な元を持たない 2 つまたはそれ以上の集合は，**分離している**あるいは**互いに素である**(disjoint)と言う．もし

$$A\cap B=\phi$$

ならば，2 つの集合は分離している．3 つ(またはそれ以上)の集合が分離しているための必要条件は，それら集合の可能な**対のすべて**が分離していることである．(**注意**：$A\cap B=\phi$ かつ $B\cap C=\phi$ であるからといって，必ずしも，$A\cap C=\phi$ あるいは A, B, C は分離しているということにはならない．A, C は重なっているが B は孤立している場合のベン図を見れば，このことは簡単にわかる．)

ある与えられた集合 X からその部分集合を作る方法は，たくさんある．部分集合の集まりの 1 つは，特に興味のあるものである．X の**分割**(partition)

とは，それらの和集合が X に等しいような X の分離している部分集合から成る任意の集合である．

例：
$$X = \{a, b, c, d, e\}$$

X の分割の可能なものの1つは，集合 $S = \{\{a, b, e\}, \{c, d\}\}$ である．このように，分割は X の元をすべて含むが，分割を構成している部分集合の間には共通な元はない．

元が n 個の集合の分割の数は B_n で表されるが，これは長い間数学者の関心を引いてきた．n が増えるに従い，B_n は**指数の数**と呼ばれる系列を構成する，B_n を与える1つの方法は，その母関数によるものである．すなわち，B_n は次の式で陰に与えられる（参考文献 RO を参照のこと）．

$$\sum_{n=0}^{\infty} \frac{B_n}{n!} x^n = e^{e^x - 1} \qquad \text{ただし } B_0 = 1$$

しかし，この公式を使って B_n の実際の値を求めることは，右辺を x のべき級数に展開し左辺と項の一致をとることを含んだ，やっかいな仕事となる．次にあげるのは，計算するのにより適した，再帰形の公式である．

$$B_{n+1} = \sum_{k=0}^{n} C(n, k) B_k$$

これの最初のいくつかの数値は：

n	0	1	2	3	4	5	6	7	8	9	10
B_n	1	1	2	5	15	52	203	877	4140	21147	115975

この表から，たとえば元が6個の集合に対し可能な分割は203種ある，すなわち元が6個の集合の部分集合を列挙する方法は203通りあるということがわかる．どの方法においても，どれかの元を共有するような2つの部分集合はなく，すべての部分集合は全体として6個の元をすべて含む．

3-5　順序対，デカルト積，写像，関数

順序対(ordered pair)は次のように表わされる：

$$(x, y)$$

これは，2つの量を一体として規定し，x が“1番目”で y が“2番目”であることを述べている．順序対 (x, y) は集合 $\{x, y\}$ と同じでは**ない**．なぜなら，

順序や位置は，集合にとっては意味がないのに対し，順序対にとっては本質的なものであるからである．したがって

$$\{x, y\} = \{y, x\}$$

であるが，一般に

$$(x, y) \neq (y, x)$$

集合の表記法のみを使って，順序対で表わすのと同じ情報を与えることができる．すなわち

$$(x, y) = \{\{x\}, \{x, y\}\}$$

小かっこを使った書き方の方が簡単であるのは明らかである．次のように，順序対を拡張して，**順序3字組**(ordered triple)が定義される：

$$(x, y, z) = ((x, y), z)$$

同じようにして，**順序 n 字組**(ordered n-tuple)も定義され，これはまた**ベクトル**とも呼ばれる．

与えられた2つの集合 A, B の**デカルト積**(cartesian product)集合は，$A \times B$ と表わされ，順序対の1番目の要素が A の元であり，2番目の要素が B の元であるような相異なるすべての順序対の集合であると定義される．したがって，2つの集合のデカルト積を作ることは，A の元のおのおのと，B の元のおのおのとの間の**対応**づけを行なうための方法となる．次に掲げるのはデカルト積の形式的な定義である：

$$A \times B = \{(x, y) | x \in A \quad \text{かつ } y \in B\}$$

例：
$$A = \{a, b, c\} \quad B = \{1, 2\}$$
$$A \times B = \{(a, 1), (a, 2), (b, 1), (b, 2), (c, 1), (c, 2)\}$$

$A \times B$ の元の個数は明らかに A の元の個数と B の元の個数との積である．このことは，簡単な推論または3-1節で与えられた積の規則を使えばわかる．

写像 (mapping) あるいは**関数**(function)は，デカルト積のある種の部分集合である．A, B を空でない集合としよう．Mを，$A \times B$ のある部分集合とすると，M は順序対を並べたものであり，それらはすべて $A \times B$ からとったものとなる．A の元のおのおのが M の順序対の1番目の要素として厳密に1つの順序対にだけ現われるならば，M は **A から B の中への写像**(mapping of A into B) と呼ばれる．この定義をもっとよく理解するためには，M の順序

対は，列が2つの表を構成しているものと考えればよい．M が **A から B の中への**写像であるならば，集合 A のすべての元は，表の左の列にただ一度だけ現われるであろう．このように，A のすべての元が完全に包含されており，しかも左の列でのそれらの唯一性も保障されることになる．離散的関数は，それゆえ2つの値の集合の間での対応関係である．左側の集合は関数の**定義域**(domain)，右側の集合は**値域**(range)と呼ばれる．定義域の元は，**引数**(argument)または**独立変数の値**と呼ばれる．値域の元は，**従属変数の値**と呼ばれる．

M が A から B の中への写像であり，B のおのおのの元が M の**少なくとも**1つの元(順序対)の第2項に現われるならば，M は **A から B の上への写像**(mapping of A onto B)と言われる．したがって**上への**写像は，**中への**写像の特別な種類であり，これによれば，写像の表による表現の2番目の列において，B のすべての元は完全に包含される(おそらく元は重複して現われる)．

例：　$A = \{a, b, c\}$　$B = \{1, 2\}$ とすれば
$A \times B = \{(a, 1), (a, 2), (b, 1), (b, 2), (c, 1), (c, 2)\}$ である．

明らかに $A \times B$ はそれ自身では A から B の中への写像では**ない**．なぜなら，たとえば b が対の左の要素として2回以上現われているからである．しかし，

$$\{(a, 1), (b, 1), (c. 1)\}$$

は A から B の**中への**写像である．しかし，集合 B のすべての元が包含されてはいない(対の右の要素に，B の元2が現われていない)ので，B の**上への**写像ではない．しかし，

$$\{(a, 1), (b, 2), (c, 2)\}$$

は，A から Bの**中への**写像であると同時に B の**上への**写像でもある．

3-6　再帰関数 [BA, KL, MC 参照]

再帰という考えの本質を形式的でない言い方で言えば，次のようになる．いかなる関数の計算も，変数の代入というむしろ一般的な方法(これは広範囲の関数について適用される方法である)とともに，少数のもっと簡単な関数の繰り返し計算を使って行なうことができる．1つの例は，順列の個数を与える関数である(3-1 節参照)．

$$P(n, 0) = 1 \tag{1}$$

$$P(n, k) = \frac{n!}{(n-k)!} \qquad 1 \leq k \leq n \tag{2}$$

ここでは，階乗，減算，除算という基本的な演算(プリミティブ)によって，関数が表わされている．われわれは通常，関数はありふれた算数のプリミティブによって定義されているはずだと考えている．上の例で，たとえば $P(5, 2)$ を計算してみると

$$P(5, 2) = \frac{5!}{(5-2)!} = \frac{5!}{3!} = \frac{5 \times 4 \times 3 \times 2 \times 1}{3 \times 2 \times 1} = 20$$

この過程では1回の減算，6回の乗算，3回の除算が必要なことがわかる．

ここで，同じ関数を計算する，代りの方法を考えてみよう．これは，簡単な代数を使って導出される．

$$P(n, 0) = 1 \qquad {}^{\dagger}(1)$$

$$P(n, k+1) = (n-k) \times P(n, k) \qquad 0 \leq k \leq n \tag{3}$$

この**再帰形**においては，関数名(P)が等号の両側に現われている．$P(5, 2)$ を例にとってみると，上の再帰方程式は，次のような代入過程の方式を記述しているものと解釈される．

$P(5, 2) = (5-1) \times P(5, 1)$　　式(3)による　　(4a)

$P(5, 1) = (5-0) \times P(5, 0)$　　式(3)による　　(4b)

式(1)により $P(5, 0) = 1$ であるので

$P(5, 1) = 5 \times 1 = 5$　　[式(4b)への後向き代入による]　　(4c)

$P(5, 2) = 4 \times 5 = 20$　　[式(4a)への後向き代入による]　　(4d)

前の方法と演算の回数を比較してみると，乗算は2回だけ必要で除算は必要でないが，代入の演算はより複雑であり，しかも減算が2回必要となっている．また，この過程は，途中の値 $P(5, 0)$ と $P(5, 1)$ を自動的に計算することにも注意しよう．代入の演算はある規則正しい順序，すなわちまず，引数の値を減らしながら関数を呼びだしていき，次いで最後に出会うものを最初に，すなわち**後入れ先出し** (last-in-first-out; **LIFO**) の順で後向き代入を行なうという順序で行なわれるので，多くの関数に対して単一の代入方式を使えばよいということが示唆されていると考えられる．プログラミングにおいては，前向

† 次の形も，これと同値である．

$$P(n, 0) = 1 \qquad (1\dagger)$$

$$P(n, k) = (n-k+1) \times P(n, k-1) \qquad 1 \leq k \leq n \qquad (3\dagger)$$

```
    ∇R←N PERM K
[1]    →(K≠0)/LAST
[2]    R←1
[3]    →0
[4]    LAST: R←(N-(K-1)) ×N PERM K-1
       ∇
```

この関数が　6PERM2
によって呼ばれると
APL プログラミング・システムは次のように実行を行なう：

For		
K=2	R←(6-1)×6 PERM 1	行 [4] 1番目のコピー（本体）
K=1	R←(6-0)×6 PERM 0	行 [4] 2番目のコピー
K=0	R←1	行 [2] 3番目のコピー
K=1	R←(6-0)×1	行 [4] 2番目のコピー
K=2	R←(6-1)×6	行 [4] 1番目のコピー
	Exit	行 [3] 1番目のコピー

ANSWER PRINTED: 30 （呼び出しの中で R← が省略されていたので）

図 3-6.1　APL システムによる 6, 2 を引数とする再帰関数 PERM の実行の様子

きの過程における引数の“積み上げ”と，後向き代入におけるそれらの使用を，**スタック**あるいはプッシュ・ダウン**記憶**であると言う．もちろん，このタイプの構造は，どんなプログラミング言語を使っても，明確な形でプログラムされるが，いくつかの言語(APL, PL/I, ALGOL が含まれる)では，言語の構文法の中に，関数を再帰的に呼び出す能力が与えられており，再帰的な呼び出しがあると，スタック参照の機構が自動的に供給される．たとえば APL では，上記の P 関数は図 3-6.1 に示すようにプログラムされる．この図はまた，行 [4] で生成される関数――この行が関数の名前 (PERM) を含んでいるのが見える――への呼び出しの系列を示している．行 [4] が最初に実行される時には，そこで見つかった PERM は，PERM プログラムの“新しいコピー”を呼出し，$K-1$ という値をそれに手渡すものと考えられる．このコピーを実行すると**自分自身の**行 [4] に到達することが当然起り得て，その時には再び PERM が呼び出され，新しいコピーが活性化される，等々．これらのコピーのおのおのは，次に**新しい**コピーを呼び出した時点で，PERM という名前のすぐ左の位置に“立ち往生”したままであることに注意しよう．引数の値が 0 のコピー

が呼び出されると，行 [1] の判定・分岐の結果，行 [2] へ進み，そこで R が 1 にセットされ，次いで行 [3] に進み，現在の PERM コピーから出て行き，以前呼出された PERM コピーへと進む．このコピーは，**自分自身の**行 [4] で戻されてきた R の値を使い，さらに前に呼び出されたコピーへと出て行く，等々である．このようにして，後向き代入が実現される．実際には，引数の値のスタックだけが記憶装置に保持されている；関数の物理的コピーは 1 つだけであり，これが参照されるべきすべての論理的コピーの役目をはたしている．

それではここで，関数の値を求める直接的な方法と再帰法との効率の比較について考えてみよう．関数 $F(X, Y)$ の値を，$Y = S, S+1, \cdots, S+N$ について計算するものとしよう．ここで，$F(X, S) = C$ が与えられており，X, C, N は定数である．2 つの基礎的なアプローチを対比しよう．

1. **直接法**　計算は，加，減，乗，除などの“普遍的な”プリミティブだけを使って行なわれ，この場合に F_i を計算するにはこれらの演算が K_{F_i} 回必要であるとすれば，

$$\sum_{i=1}^{N} K_{F_i}$$

回の演算が，N 個の値を計算するのに必要となる．

2. **再帰法**　計算は，次の方程式によって行なわれる．

$$F(X, S) = C \tag{5}$$

$$F(X, Y+1) = G(X, Y, F(X, Y)) \tag{6}$$

これは，新しい関数 G を発見するための解析を必要とする．G の引数は，X, Y および F の**前の**値である．G_i を計算するのに K_{G_i} 回のプリミティブ演算が必要だとすれば，F の値を N 個求めるには

$$\sum_{i=1}^{N} K_{G_i}$$

回のプリミティブ演算(および代入演算)が必要となる．

しばしば見られるように，$\sum K_{G_i}$ が $\sum K_{Fi}$ より小さい場合には，間接法の方が効率がよい(少なくとも，効率の尺度として演算回数を用いた限りにおいて)．演算回数が改善されるのは，次のような事実から生じてくるものと考えられる．すなわち，おのおのの関数の値を計算するために普遍的なプリミティブに立ち戻るかわりに，再帰法は，関数それ自身をプリミティブとして用いている

のである！　このようにして，関数の前の値には，独特の複雑さが含まれているが，次の値を計算するのに利用できるのである．関数 F が与えられた時に関数 G を見つけるための“お料理読本”的な方法は，一般的にはないのだということに注意しよう［式(6)参照］．これには主として工夫の才が必要となるのである．

3-7　確率の概念：標本空間と公理［FE, FR, MO 参照］

確率論のはじまりは，少くとも，17世紀に行なわれた，ある運まかせのゲームの勝ち目を解析する試みにまでさかのぼる．19世紀には，Gibbs, Maxwell, Boltzman などの仕事を通じて，物理学の多くが確率モデルに向かって進んで行った．これは20世紀に入っても引続き発展し，M. Planck と E. Schrödinger による量子論にまで及んだ．

システム科学やエンジニアリングもまた確率を大いに利用している．Shannon の情報理論や，電話のシステムを設計するために A. K. Erlang が1909年に初めて導入した待ち行列モデルも，すべて確率を基礎としている（第**5**章参照）．

統計学はデータの解析と記述とに関与している．確率論は，データのある特性の出現を説明できるような，過程のある種の抽象を取り扱う．このようなどちらかといえば莫然とした言い方でさえ，**2**つの学問が密接に関係していることを示している．なぜなら，われわれは通常，データの量的な記述だけではなく，観察されたデータを生成することができるようなシステムモデルをも捜し求めるからである．一旦利用できるとなれば，そのようなモデルは，システム・パラメータの間の関係を調べるのに用いることができる．近代統計学の多くは，ある仮定されたモデルにデータがうまくあうかどうかを決めるために，データを解析することに関与している．このように，統計学と確率論とは密接に関係しているのである．

確率論は，たいていの理論と同様，現実世界そのものでは**なく**，その**モデル**を取り扱う．どんなモデルも完全に包括的ではなく，あるかなり限定された目的にのみ役立つ．たとえば，タイムシェアリング・システムの設計においては，測定により支えられた次のような仮定は，しばしばよい仮定となる．すなわち，ユーザは統計上からみれば似かよっている，言いかえれば，おのおのの微小時間間隔において，まだサービスを受けていないユーザのおのおのは，おそらく他のユーザと同じように要求を発するであろうという仮定である．この仮定は，すべてのユーザが，システムの利用のすべての局面で同じようにふるま

うとは**言っていない**．異なる顧客の間では，彼らが使用する主記憶の大きさや処理時間などのようないろいろな特性は，実際には相異なっている．それでもなお，起りうる要求時間が同じであるという仮定は良い仮定であり，これによって，解析的モデル(特にこのモデルは第5章で詳細にとり扱われる)の公式化が，大いに簡単になる．

われわれの関する限りにおいて，確率論には2つの側面がある：(1) モデルの構築と解に非常に必須である推論の方法論と手段，(2) 現実世界の多くの状況にあてはまることがわかっている，二項，Poisson, 正規などのいくつかのモデルの解析結果．このうち前者の方が，より基礎的であり，最終的にはより重要である．後に，いくつかの有名なモデルを研究する過程で多くの例が出てくるが，これらのいくつかはそれ自身大切なものであり，またこれらすべては，モデルの作成と解の実例として重要である．

本書では，この章で離散型の確率変数の確率論について議論し，後の章でこれを使う．連続型の確率論の基礎についての簡単な説明は，補遺IIに与えられている．

根元事象(elementary event)と呼ばれる，有限個の相異なる結果を伴なった試行を考えよう．それぞれの根元事象は，それら独自のラベルによって識別される．可能な結果のアルファベットを構成している，このラベルの**集合**をその試行の**標本空間**(sample space)または**状態空間**(state space)と呼ぶ．

例：

1. “硬貨を1回投げる”という試行に対しては，標本空間は，**表**(head)と**裏**(tail)という2つの点を持ち，したがって，空間は次の集合となる．

$$X = \{h, t\}$$

2. “硬貨を3回投げる”という試行に対しては，おのおのの標本点は，3つの記号の系列で都合のよいようにラベルが付けられる．したがって，“1回目が表，2回目も表，3回目が裏”という根元事象には hht というラベルが付けられ，全標本空間は次のようになる：

$$X = \{hhh, hht, hth, htt, thh, tht, tth, ttt\}$$

事象(event)**とは標本点の部分集合である**．したがって“裏が2回”という事象は X の部分集合 $\{htt, tht, tth\}$ である．確率論は事象に関わりを持っているので，集合論で用いる言語が強力な道具となる．確率の問題における普遍集

合は，全標本空間である．硬貨投げの例で見られたように，標本点のラベルは，数字である必要はない．しかしこれらは，相異なっており，しかも，全体として確率モデルの可能な試行結果のすべてを説明するものでなければならない．

確率(probability)とは，標本空間 X の各点 x あるいは事象 A(点の集合)に割当てられた数値である．$P(A)$ と表わされる事象 A の確率は，次の公理を満足しなければならない．

A1：範　囲　　$0 \leq P(A) \leq 1$　確率の範囲は，0から1まで(両端を含む)の正の数である．

A2：確実性　　$P(X) = 1$　全標本空間の確率は1である．この公理は A1 とともに，確率の値を正規化している．

A3：和　　A と B とが互いに排反する(素な)事象である，すなわちそれらが共有点を持たない($A \cap B = \phi$)ならば，$P(A \cup B) = P(A) + P(B)$ である．

和集合と共通部分の結合則および分配則を使って，公理 A3 は3つあるいはそれ以上の集合にも拡張される．この証明は練習問題として残しておく．結果だけを述べると：

標本空間の互いに素な3つの部分集合を S_1, S_2, S_3 と表わす．すなわち

$$S_1 \cap S_2 = \phi \quad \text{かつ} \quad S_2 \cap S_3 = \phi \quad \text{かつ} \quad S_1 \cap S_3 = \phi$$

であるとすれば，

$$P(S_1 \cup S_2 \cup S_3) = P(S_1) + P(S_2) + P(S_3)$$

特別なものではあるが重要な例として，おのおのの S_i は1つの標本点 X_i から成るものと考えよう．これらはすべて相異なっているので，互いに素である．標本空間の全点の確率は，したがって標本点の確率の総和であり，A2 によってこの和は1となる．

$$P(X) = \sum_{i=1}^{n} P(X_i) = 1 \tag{1}$$

A と B とが必ずしも互いに素でないならば，A と B の和集合の確率は

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \tag{2}$$

この式は，2つの集合の和集合に対する元の個数の数えあげに似かよっているのがわかる(第3-3節参照)．ここでもまた，最後の項は，A と B の共通部

分での数えあげの重複を訂正するためのものである．空事象の確率は0である．

$$P(\phi)=0 \tag{3}$$

標本空間を構成することは，試行または過程の確率モデルを公式化するうえでのきわめて重要な一歩である．非常に複雑な現実世界の現象から，最も巧妙な抽象化が行なわれて，ときには数学的な解法が許される記号によるより簡単な構造が作られるのはここである．この手続きのための確固とした規則はない．場合場合に応じた判断と工夫が必要である．標本空間の構成には，次の2つの側面がある：

1. 標本点の数えあげとラベル付け
2. 各標本点への確率の割付け

読者には，公理を満足するように行なわれねばならないこれら2つの仕事に視点を置きながら，この章の残りを勉強されることをお勧めする．

標本点あるいは事象は，数値で特徴づけられる必要はないけれども，関心のあるたいていの場合には，“n 回の試行における成功の回数”というような事象は1つの数値であり，このとき**確率変数**のとる値としては，この数値が選ばれる．確率変数のおのおのの値には対応する確率があるので，これからわれわれは，確率変数およびその**確率関数**(probability function)あるいは**分布**について話をしよう．したがって，記号の対$(X, P(X))$によって，確率変数 X の値およびその分布関数 $P(X)$の集合を表わす．

3-8　独立事象：積の原理

2つの事象 A, B が**独立である**というための必要十分条件は

$$P(A\cap B)=P(A)P(B) \tag{1}$$

である．独立な事象は，互いに排反する事象では**ない**．実際，次の推論でわかるように，互いに排反する事象は，独立とはなり得ない：もし，A と B が互いに排反するならば，定義により $A\cap B=\phi$ であり，したがって $P(A\cap B)=0$ である．しかし，もし A と B が独立で，しかも $P(A)$ と $P(B)$はともに0でないならば，式(1)によって $P(A\cap B)\neq 0$ である．それゆえ，事象 A と B が互いに排反し，しかも独立であるということは起り得ない．

ある試行の標本空間とそれに結びつけられた確率とが，対$(X, P(X))$によっ

て記号で表わされているものとしよう．独立な試行$(Y, P(Y))$もまた考えてみよう．ここで“独立な”というのは，直観的には，どちらの試行ももう一方の試行から影響を受けないという意味である．さてここで，もう1つの別の試行を定義したとしよう．この試行の結果のおのおのは，X, Y の結果の特定な**対**である，すなわち，対の1番目は X から，2番目は Y からのものである．したがって，新しい標本空間 Z には，デカルト積によってラベルを付けることができる．

$$Z = X \times Y \tag{2}$$

この新しい標本空間内の各点の確率は，その点を構成する X, Y 事象の確率の積である．このような性質は**積の原理**(product principle)と呼ばれる．Z とその確率が公理を満たすことを示すのは，困難なことではない．

例：

1. “くせのない”硬貨を1回投げることは，次のような標本空間と確率を持つ．

$$X = \{h, t\} \qquad 1/2, 1/2$$

2. “くせのない”さいころを1回ふることは，次のような標本空間と確率を持つ．

$$Y = \{1, 2, 3, 4, 5, 6\} \qquad 1/6, 1/6, 1/6, 1/6, 1/6, 1/6$$

3. “硬貨を投げ，それからサイコロをふる”という試行は，次のような標本点と確率を持つ．

$$Z = \{h1, h2, h3, h4, h5, h6, t1, t2, t3, t4, t5, t6\}$$
$$p = 1/12, 1/12, \cdots, 1/12$$

いくつかの場合には，標本空間のデカルト積を使って試行を簡単に特徴づけることができない，というのは，標本点のいろいろなクラスに対し，根元事象の確率の積の適用の仕方が異なっていなければならないからである．

例：　くせのない硬貨を，表が2回出るまで，最大5回まで投げるものとする．標本空間と，その確率を求めよ．

解：　標本空間は，試行結果のいろいろな確率を示すトリー図を使って，数えあげることができる(**図3-8.1参照**)．トリーは2つの根(1回目の硬貨投

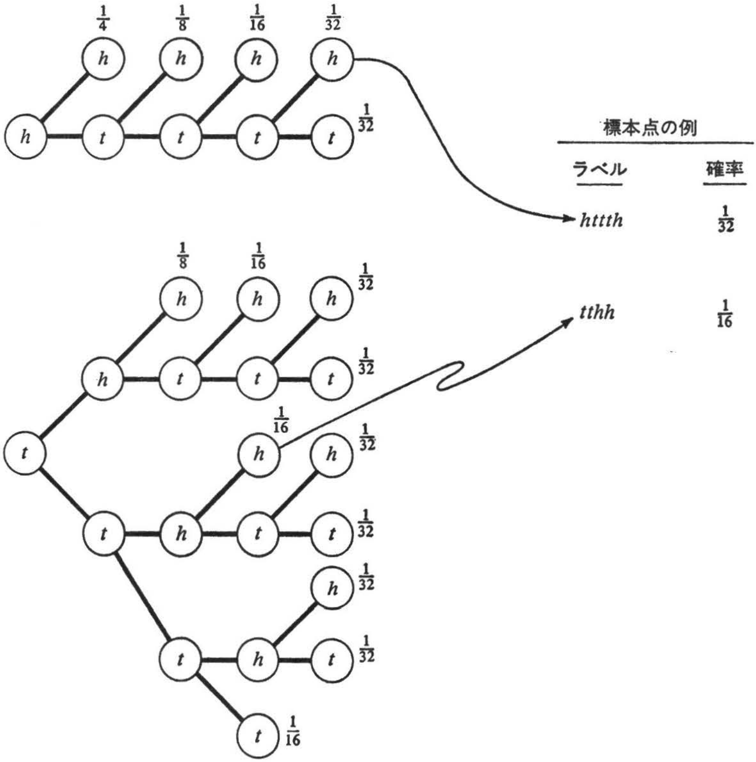


図 3-8.1　表が2回出るまで硬貨を投げたときの標本空間：投げる回数は5回までとする．

げが表か裏か）と，表が2回現われるまでの可能な硬貨投げの可能性に応じた枝とを持つ．一旦作り上げられると，標本点のラベルは，トリーの根から葉への道筋に対応するノード・ラベルを記録することによって示される．おのおののノードは，硬貨を1回投げたことを表わしているので，それぞれは1/2の確率に相当する．したがって，標本点の確率は，トリー上のその道筋に含まれるノードの数だけ1/2を掛けあわせたものとなる．

3-9　等確率標本空間

いくつかの標本空間では，標本点が**等確率**である（すなわち，すべての i について $p_i = p$）ことがわかっている，あるいは仮定できる．確率の全総和は1であるので，

$$P(X) = \sum_{i=1}^{n} p_i = np = 1 \qquad (1)$$

したがって

$$p = \frac{1}{n} \qquad (2)$$

事象は標本点のある部分集合であるから，n 個の元から成る**等確率標本空間**(equiprobable sample space)の中で元を j 個含むような事象は，j/n の確率を持つ．言いかえると：

$$P(A) = \frac{A\ \text{の元の個数}}{X\ \text{の元の個数}} \qquad (3)$$

あるいは

$$P(A) = \frac{A\ \text{が起り得る場合の数}}{X\ \text{が起り得る場合の数}} \qquad (4)$$

これらの簡単な関係が成り立つのは，等確率標本空間だけであるということを，強調しておかねばならない．

等確率標本空間の，解析に非常に便利な1つの特徴は，次に述べることである．すなわち，事象の確率は単に事象の集合内の標本点の数をかぞえることにより計算され，しかも，各点には記号系列によるラベルが付けられているので，その問題は，ある与えられた条件を満たす記号系列を数えあげる問題となる．これはまさに，組合せ解析において考えられた種類の問題である．

例：　誕生日問題

ある部屋に n 人の人間が居るものとする($n \leq 365$)．

仮定：1．誕生日の出現可能性は，すべて同じである．

2．1年は365日である．

問題：　すべての誕生日が相異なる確率を求めよ．

解：　ここでの根元事象は，n 個の誕生日の，1つの可能な集まりから成る，すなわち，それぞれ $r = 365$ 個の記号のアルファベットから選ばれた記号を n 個並べた系列である．したがって，標本空間は r^n 個の点を持ち，そのおのおのの確率は $1/r^n$ である．関心のある事象は，そのラベルにはどんな記号の重複もないような点の集合である．そのような系列の個数は，r 個の記号のアルファベットから重複なしに n 個の記号をとり出して作る

順列の個数，すなわち，$P(r, n)$である．したがって，われわれが今考えている事象の確率は，この多くの標本点の確率である．すなわち

$$p = P(r, n)\frac{1}{r^n} = \frac{r!}{(r-n)!\,r^n}$$

例: ある計算機システムは，4台のディスク駆動装置を持っており，この特定の問題においては，おのおのの駆動装置が参照される可能性は，他と同じであると仮定しよう．参照を3回行なうときの，次の事象の確率を求めよ:

1. すべての参照が1つの駆動装置に対して行なわれる．
2. すべての参照が相異なる駆動装置に対して行なわれる．
3. ちょうど2回の参照が同じ駆動装置に対して行なわれる．

解: 1回の参照の，標本点と確率は

$$A,\ B,\ C,\ D \qquad 1/4,\ 1/4,\ 1/4,\ 1/4$$

3回の参照に対しては，標本空間は $4^3 = 64$ 個の標本点から成り，これらは，それぞれ $1/4^3 = 1/64$ の確率を持ち，次のようにラベルが付けられる．

$$AAA,\ AAB,\ \cdots,\ DDD$$

上の例の **1**. で規定されている事象は，4つの標本点(AAA, BBB, CCC, DDD)から構成されている．したがって事象の確率は

$$\text{確率} = 4/64$$

2. で規定されている事象は，記号が4個のアルファベットから，記号の重複なしに，3個の記号の系列をとり出す方法の数だけ，標本点の数をかぞえることに相当する．

$$P(4, 3) = \frac{4!}{(4-3)!} = 24 \quad \text{標本点}$$

したがって，確率 $= 24/64$ である．

3. で規定されている事象の確率は，前の2つにくらべて計算するのが多少むずかしい．ここでは，“同じ”記号を2つ，それと“異なる”記号を1つ含んでいる標本点のラベルは，次にあげるうちのどれか1つの形をしている，ということを認めることから出発しよう．

$$XXY \quad XYX \quad YXX$$

これらのおのおのについて，Xを選ぶ方法は4通り，次いでYを選ぶ方法は残り3通りある．したがってこのような標本点の個数は$3 \times 4 \times 3 = 36$となり，確率は

$$\text{確率} = 36/64$$

この問題の1, 2, 3は，標本空間の**分割**を構成している(すべての点が含まれ，事象の間で点の重複がない)ということに注意しよう．この理由により，これらの事象の確率の和は1となる．

3-10　標本空間の階層

標本空間Xと事象の集合Sがあり，Sのおのおのの事象S_iはXの部分集合であるとしよう．もしSがXの**分割**である(すなわち，S_iはすべて互いに素であるが，それらの和集合はXに等しい)ならば，われわれはおのおのの**事象**S_iを，新しい標本空間S内の標本点とみなすことができる．読者は自力で，$P(X)$が公理を満たすなら$P(S)$も公理を満たすことを確かめられたい．この性質は非常に有用である，というのは，関心のある事象の確率は直接には求められないが，それと関連したより簡単な試行の確率はわかっているという場合がしばしばあるからである．もし，関心のある事象に対する標本空間を，もっと基本的な試行の互いに素な部分集合によって形成できるならば，これを使って，求めたい事象の確率を計算できる(通常，公理3またはそれを一般化したものを用いる)．

例：　“硬貨を3回投げる”という事象に対する標本空間

$$X = \{hhh,\ hht,\ hth,\ htt,\ thh,\ tht,\ tth,\ ttt\}$$

があり，すべての点は1/8の確率を持つとする．

“表の出る回数が0回，1回，2回，3回”という事象($0h$, $1h$, $2h$, $3h$とラベルを付ける)の標本空間とそれぞれの確率を求めよ．

解：　まず，問題となっている事象を集合の形で列挙する．

$$0h = \{ttt\} \quad 1h = \{htt,\ tht,\ tth\} \quad 2h = \{hht,\ hth,\ thh\} \quad 3h = \{hhh\}$$

これを見ると，これらの集合は互いに素であり，またこれらの和集合はもとの標本空間Xであることがわかる．したがって，次のような，有効な新しい標本空間を考えることができる．

$$H = \{0h,\ 1h,\ 2h,\ 3h\}$$

それでは，公理3を使ってこれらの事象の確率を計算しよう†.

$P(0h) = 1/8$　なんとならば，与えられた通り $P(ttt) = 1/8$ である.

$$\begin{aligned} P(1h) &= P(htt,\ tht,\ tth) = P(htt \cup tht \cup tth) \\ &= P(htt) + P(tht) + P(tth) \\ &= 1/8 + 1/8 + 1/8 = 3/8 \end{aligned}$$

$P(2h) = 3/8$　($P(1h)$と同様に)

$P(3h) = 1/8$　($P(0h)$と同様に)

他の標本空間から新しい標本空間を作り出すという考えは，次の例に示すように，しばしば，いくつかのレベルへと敷衍することができる.

例:　硬貨を3回投げるという事象に対する標本空間 X が与えられている(上を見よ). **表が2回以上でる**確率を求めよ.

解:　この新しい問題の第一段階は，標本空間 H とその確率を求めた前の例題そのものである. 次に，この標本空間を使って，問題となっている事象を含んでいる新しい標本空間を数えあげる.

$$\{h \geq 2\} = (2h,\ 3h\}$$
$$\{h < 2\} = \{0h,\ 1h\}$$

問題となっている事象は，これの最初のものであるから:

$$P(h \geq 2) = P(2h \cup 3\,h)$$

公理3により:

$$P(h \geq 2) = P(2h) + P(3h)$$

前の例題の確率を使って

$$P(h \geq 2) = \frac{3}{8} + \frac{1}{8} = \frac{4}{8} \quad \textbf{(答)}$$

ときには，最後の例で見られたように，ある特定の問題を解くためには，標本空間のすべての要素を列挙する必要がないことがある [すなわち，$P(h < 2)$ のものを列挙することには，直接の興味がない]. しかし，初めて確率を学ぶ

† $P(\cdots)$ というように，確率の引数になっている集合を表記する際，集合を並べるときの区切りとしてふつう使う中かっこを省略することにする.

ときには特に，標本点と空間のすべてを知っておくことが，しばしば大切である．練習をつんでもっと慣れてくると，多くの近道を見つけることができる．

3-11　条件つき確率

点が n 個ある標本空間 X が与えられており，その各点には次の式を満足する確率が与えられているものとする．

$$\sum_{i=1}^{n} p_i = 1 \tag{1}$$

X の最初の k 個の標本点から構成される事象 B に焦点をあててみよう（このように，われわれはいつでも点を数えることができるとする）．

$$P(B) = \sum_{i=1}^{k} p_i \tag{2}$$

また，その点のいくつかが事象 B と相交わっているような事象 A を考える．さて，われわれは，事象 B が実際に起ったということを知っていると仮定しよう．ここで生じる質問は次のものである：$P(A|B)$ と書かれ，B が起ったという条件下での A の確率を意味する，事象 A の**新しい**確率とは何であろうか．B がすでに起ったことを知るということは，B のすべての要素だけを含んだ標本空間を新たに作るということに等しい．この B はやはり k 個の標本点から構成されているけれども，いまや p_i' と呼ばれるこれらの点の確率は，すべての標本空間についての公理

$$\sum_{i=1}^{k} p_i' = 1 \tag{3}$$

を満足しなければならない．p_i' の互いの比は，p_i のそれと同じはずである，すなわち，p_i' と p_i はある定数 ν によって

$$p_i' = \nu p_i \tag{4}$$

の関係にあると考えることは，理にかなったことと思われる．この関係を式(3)に代入し，式(2)を使えば：

$$\nu = \frac{1}{\sum_{i=1}^{k} p_i} = \frac{1}{P(B)} \tag{5}$$

このように，B の各要素に今や結びつけられる確率は，B が起ったということ

をわれわれが知る前の確率 (p) に，係数 $1/P(B)$ を乗じたものである．$(A|B)$ と呼ばれる，A に対応する事象は，A に属しまた B にも属するような標本点の集合である：すなわち，この事象は，集合 $A \cap B$ である．したがって，この事象の確率は，$P(A \cap B)$に係数 $1/P(B)$を乗じたものとなる．それゆえ，次の公式を得る．

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{6}$$

この公式は，$P(B)$が 0 でない，すなわち B が空事象でない場合に限り，成りたつ．

例：　ある契約に入札している 4 つの計算機会社 A, B, C, D がある．これらの会社が過去に同じような契約に入札した例を調べた結果，落札に成功した確率は，次のようになることがわかった．

$$P(A) = 0.3,\ P(B) = 0.2,\ P(C) = 0.4,\ P(D) = 0.1$$

契約決定が行なわれるまえに，B 会社は入札をとり下げたとする．A, C, D が落札に成功する新しい確率を求めよ．

解：　"B が入札をとり下げた" という事象に Y とラベルをつけ，これはまた "B が落札に成功し**ない**" という事象であることに気づくと $P(Y) = 1 - P(B) = 0.8$ である．A, C, D はすべて Y の部分集合であるから，集合 $A \cap Y$, $C \cap Y$, $D \cap Y$ はそれぞれ集合 A, C, D に等しい．したがって式(6)から

$$P(A|Y) = \frac{P(A)}{P(Y)} = \frac{0.3}{0.8} = \frac{3}{8}$$

$$P(C|Y) = \frac{P(C)}{P(Y)} = \frac{0.4}{0.8} = \frac{4}{8}$$

$$P(D|Y) = \frac{P(D)}{P(Y)} = \frac{0.1}{0.8} = \frac{1}{8}$$

条件つき確率の原理を一般化したものは次の定理となる：

Bayes の定理　標本空間 X の n 個への分割 S が与えられている．すなわち，n 個の事象 S_i はすべて分離しており，それらの和集合は X に等しいものとする．確率 $P(S_i)$はしばしば，事象 S_i の "事前確率" または先験確率と呼ばれる．もしその後である事象 E が起ったという情報を得たならば，S_i の新

しい確率(または事後確率)$P(S_i|E)$は次の式で与えられる：

$$P(S_i|E) = \frac{P(S_i \cap E)}{P(S_1 \cap E) + P(S_2 \cap E) + \cdots + P(S_n \cap E)}$$

例：　ある会社は A, B, C 3つのプラントを持っており，総生産高に対する生産高はそれぞれ 20, 30, 50% である．製品に欠陥のない確率は，それぞれ，0.85, 0.90, 0.95 である．ある顧客が欠陥製品を受けとった．これがプラント C で生産されたものである確率はいくらか．

解：　E を欠陥製品とする．$E = AE$ は，プラント A で生産された欠陥製品である(BE, CE も同様である)．したがって，$P(AE) = 0.2(1 - 0.85) = 0.03$; $P(BE) = 0.3(0.1) = 0.03$; $P(CE) = 0.5(0.05) = 0.025$ となる．これらは積の規則を使って得られる．なぜなら，欠陥の原因と，生産高の確率とは独立だからである．Bayes の定理により：

$$P(C|E) = \frac{P(CE)}{P(AE) + P(BE) + P(CE)} = \frac{25}{85} = \frac{5}{17}$$

3-12　度数比：大数の法則

確率というものの意味に対する1つの洞察は，次の実験に代表されるような状況から得られる．おのおのの実験的な試行について，たとえば $10 \leq x \leq 60$ の範囲にあることがわかっている整数 x を作り出すようなある過程（その機構はわかっていなくてもよい)を考えよう．この実験の標本空間はしたがって(60 − 10) + 1 = 51 個の標本点を持っている．そこで，この実験がたとえば 10 回繰り返されたとすれば，典型的なものとして次の 10 個の結果が観察されるかもしれない．

39, 41, 20, 40, 41, 12, 36, 60, 41, 38

この系列を荒っぽく調べただけでも，たいていの値が 40(または 41)のあたりにかたまる傾向があることがわかるが，しかし，20, 12, 60 などの 40 からまったく離れた値もいくつかある．いずれにせよ，数値を作り出すこの過程は，試行の結果が 40 に“集中する 傾向にある”（40 から離れた 値をとる 傾向は小さい）ということを示しているように思われる．問題はこの直観的な考えを正確に記述することである．この目的のために，どんな試行系列に対しても，標本点のすべてとおのおのの出現度数，ならびに(実験的試行の)数値の総出現度数に対するおのおのの出現度数の比を並べあげた度数分布表(第2章参照)を作

ることができる.

この実験を10回よりもっと多く繰り返したなら，(実験がすべてお互いに独立ならば) 数値を生成しつつある，基礎となっている(しかも未知の)過程は，次のような一貫性を示すことがおそらく期待される：おのおのの起こり得る結果は，試行を10回行なったときより100回行なったときの方が数多く出現するし，100回よりも1000回行なったときの方がもっと多く出現するものと期待されるが，全試行回数に対するどんな特定の結果の出現回数の**比率**も，試行回数が増えるにつれて，ある極限値に近づく傾向にある。このことを正確に言うため，m 回の試行の連続における x の出現回数を $N_m(x)$ で表わそう．$P_m(x)$ を次のように定義する.

$$P_m(x) = \frac{N_m(x)}{m} = \frac{\text{結果 } x \text{ の出現回数}}{\text{試行回数}}$$

m が無限に大きくなったときの比 $P_m(x)$ が，結果 x の確率の1つの解釈である．この比がある極限値 p に近づくであろうという確信は，多くの(しかしすべてではない)観察可能な現象にもとづいた**確信**である．この性質はまた，**大数の法則**(law of large numbers) と呼ばれ，数学的には次のように表現される.

どんなに小さな $\epsilon > 0$ が与えられても

$$\underset{m\to\infty}{\text{確率}}\left(\left|\frac{N_m}{m} - p\right| < \epsilon\right) = 1$$

大数の法則は，現実世界の法則では**なく**むしろ，抽象的ではあるけれども非常に多様なシステムにあてはまる (それらのふるまいを観察することによって判定される) と考えられるモデルの性質と考えるべきである．ある数学的な確率モデルに対して，上の式を厳密に証明することもできる(第3-14節参照).

大数の法則を注意して読むと，これは，m が大きくなるにつれて度数比 N_m/m は確率 p に近づくと主張しているのでは**なく**，この比が p に近いという**確率**が非常に1に近いと主張していることがわかる．したがって，m が非常に大きくても，度数比が p と非常に異なるという機会が常にある.

3-13 期待値と分散

統計学(2章)で使われているのとまったく類似の方法で，またまったく同じ理由で，確率変数を記述する数値を求める．この概念は，統計学において考えられたものと似ているので，われわれは確率変数に対してのいくつかの結果だ

けを列記する．しかしながら，確率変数の場合には，総和は，たいてい**標本点の値**についてとるのであり，実際の試行系列についての総和をとるのではないということに注意しておくのはむだではない．確率変数の値を X_j と表わせば公理によって

$$\sum_{j=1}^{n} P(X_j) = 1$$

期待値(平均値)

$$\boldsymbol{E}(\boldsymbol{X}) = \boldsymbol{\mu} = \sum_{j=1}^{n} \boldsymbol{X}_j \boldsymbol{P}(\boldsymbol{X}_j) \tag{1}$$

このように，確率変数の期待値はその標本点の値の重みづき和であり，各点にはその確率によって重みがつけられている．

さて，m 個の確率変数 $X_1, X_2, \cdots, X_m$ を考えるものとする．添字の使い方が式(**1**)と違っていることに注意しよう．式(**1**)では X_j は標本空間内の**1**つの値を意味しているが，こんどは**1**つの確率変数，すなわち，空間 j についての標本値の全集合を意味している．もし変数が期待値 $\mu_1, \mu_2, \cdots, \mu_m$ を持つならば，和の期待値に関する定理は次のようになる．

定理 3-13.1

$$E(X_1 + X_2 + \cdots + X_m) = E(X_1) + E(X_2) + \cdots + E(X_m) = \sum_{i=1}^{m} \mu_i \tag{2}$$

これの重要な特例は，変数のすべてが同一の平均値 μ_1 を持つ場合で，この場合には

$$E(mX) = m\mu_1 \tag{3}$$

確率変数 X の**分散**は

$$\begin{aligned} V(X) &= \sigma^2 = E((X - E(X))^2) \\ &= \sum_{j=1}^{n} (X_j - \mu)^2 P(X_j) = \left(\sum_{j=1}^{n} X_j^2 P(X_j)\right) - \mu^2 \end{aligned} \tag{4}$$

標準偏差 σ は分散の平方根である．

いくつかの確率変数の和から成る確率変数の分散を考えた場合，これに適用できる定理(ここでは証明なしに述べるが，参考文献**1**を参照されたい)は，**定理 3-13.1** の姉妹定理とも言うべき次の定理である．

定理 3-13.2 和の分数 $X_1, X_2, \cdots, X_m$ は，それぞれ $\sigma_1{}^2, \sigma_2{}^2, \cdots, \sigma_m{}^2$ の分散を持つ m 個の確率変数とし，S_m は確率変数の和 $S_m = X_1 + X_2 + \cdots + X_m$ であるとする．このとき，S_m の分散は：

$$V(S_m) = \sum_{i=1}^{m} \sigma_i{}^2 + 2\sum_{j,k} \mathrm{COV}(X_j, X_k) \tag{5}$$

ここで，最後の和は，可能な $C(m, 2)$個の，$j \neq k$ なる対(X_j, X_k)のそれぞれをただ**1**つずつだけ含んだものである．

重要な特例は，変数 X_j が**互いに独立**な場合であり，その場合共分散はすべて**0**となって：

$$V(S_m) = \sigma_1{}^2 + \sigma_2{}^2 + \cdots + \sigma_m{}^2 \tag{6}$$

さらに，すべての分散が，たとえば $\sigma_1{}^2$ に等しい場合には：

$$V(S_m) = m\sigma_1{}^2 \tag{7a}$$

したがって：

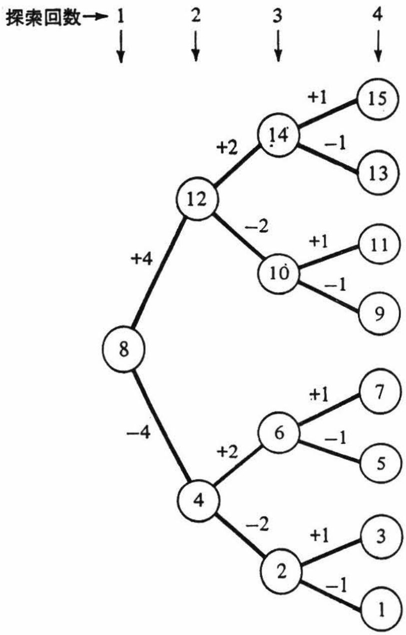
$$\sigma_{S_m} = \sqrt{m}\,\sigma_1 \tag{7b}$$

上記の定理の応用の**1**つは，**2分探索法**(binary search；**図 3-13.1**)における探索の平均回数を求める公式を導出することである．**2**分探索法は，相異なる(重複がない)数値の**ソートずみリスト**(ファイル)X を効率よく探索して，X 内のある与えられた引数の値 A に合致する(等しい)要素の位置を示す数字を見つける技法である．

ここで，X 内の要素の個数 n は，**2**の整数乗より**1**少ない数であると仮定しよう．

$$n = 2^m - 1 \qquad \text{あるいは} \qquad m = \log_2(n+1)$$

探索アルゴリズムは，X 内でのインデクスの値をつぎつぎと生成し，その最初のインデクスはファイルの中央にある．引続くインデクス I のおのおのは A と $X[I]$ とを比較するのに使われる．両者が等しいと，探索が完了する．そうでない場合，$X[I] > A$ であれば，I に正の増分を加え，$X[I] < A$ であれば，I に負の増加を加える．それぞれの増分の大きさは，**1**つ前の増分の半分の大きさである．それから新しい I を使って次の探索が行なわれる，等々．この過程は，**すべて**の可能な I の値をノードとするトリー図を使って，図に表わされる(図 **3-13.1** 参照．これは $m = 4$ の場合である)．ノードを結ぶ線のそれぞれに



$n = 15;\ m = 4$
増分の大きさ：　8, 4, 2, 1

探索回数 (k)	到達可能なセルの数	セルの確率	積
1	1	$\frac{1}{15}$	$\frac{1}{15}$
2	2	$\frac{1}{15}$	$\frac{4}{15}$
3	4	$\frac{1}{15}$	$\frac{12}{15}$
4	8	$\frac{1}{15}$	$\frac{32}{15}$
k	2^{k-1}	$\frac{1}{15}$	$k2^{k-1}(1/n)$

$$E(k) = (\tfrac{1}{15})(1 + 4 + 12 + 32)$$
$$= \tfrac{49}{15} = 3.27$$

図 3-13.1　2分探索法

は，現在のインデクスから次のインデクスを求めるのに使われる正または負の増分によってラベルがつけられている．

探索が完了するまでの探索の平均回数を求める公式を導出するために，X 内の項目はすべて確率が等しいと仮定しよう．X のいろいろな項目に到達するのに必要な探索回数は，表を用いて表わされるであろう．このように，k 回目の探索では，2^{k-1} 個の項目に到達することができる．探索の平均回数は，したがって：

$$E(k) = \sum_{k=1}^{m=\log_2(n+1)} k\,2^{k-1}\left(\frac{1}{n}\right)$$

右辺の式の中心部分は，次のように表わされる

$$\sum_{k=1}^{j} k\,2^{k} = 2(1 + (j-1)2^{j})$$

これと少々の代数を使えば:

$$E(k) = \left(\left(\frac{n+1}{n}\right)\log_2(n+1)\right) - 1$$

探索するリストが大きいときには，$n \gg 1$ であり，したがって:

$$E(k) \fallingdotseq (\log_2 n) - 1$$

このように，探索の平均回数は最大値より 1 だけ少ない数となる．

3-14 Markov と Chebyshev の不等式

これらの重要な一般公式は，平均値と分散によって，確率の限界値を与える．これらは，形のうえから特定の確率関数には無関係であり，実用上でも，また確率論の定理を証明するうえでも役に立つ．

点 x(ここでは非負の数と仮定する)をもつ標本空間が与えられている．このとき t を任意の正の数とすると，期待値は次のように書かれる．

$$E(X) = \sum_x xP(x) = \sum_{x<t} xP(x) + \sum_{x \geq t} xP(x) \tag{1}$$

ここでは，すべての x の値についての総和がとられている．x はすべて非負なので，右辺の 2 つの総和はともに非負である．2 番目の総和だけを残すと，明らかにこの大きさはたかだか $E(X)$ であり，したがって:

$$E(X) \geq \sum_{x \geq t} xP(x) \tag{2}$$

$x \geq t$ で総和をとるので，すべての x をそれより小さな値 t と置きかえても不等式はなりたつ．

$$E(X) \geq \sum_{x \geq t} tP(x) = t\sum_{x \geq t} P(x) = tP(x \geq t) \tag{3}$$

この右辺の形においては，標本空間の，P によって参照される部分を示すのに，集合を簡潔に表わす記法を使っている．式(3)を変形すると，**Markov の不等式**を得る．

$$P(x \geq t) \leq \frac{E(X)}{t} \tag{4}$$

例: ある実験を 1000 回行なったときの成功回数の期待値が 420 であるとする．800 回以上成功する確率はしたがって**たかだか**

$$P(x \geq t) \leq \frac{420}{800} = 0.525$$

である．

X を $(X-E(X))^2$ で置きかえ，t を t^2 で置きかえると，$E(X-E(X))^2$ は分散であるので，Markov の不等式は **Chebyshev の不等式**になる．

$$P((X-E(X))^2 \geq t^2) \leq \frac{V(X)}{t^2} \tag{5}$$

ここで，$V(X)$ は X の分散である．これの別の形は：

$$P(|X-E(X)| \geq t) \leq \frac{V(X)}{t^2} \tag{6}$$

ここで，$|X-E(X)|$ は差の絶対値である．

Chebyshev の不等式の 1 つの重要な使用例は，第 3-12 節で述べた"大数の法則"を証明するために使う場合であり，ここにその証明を示す．

定理 3-14.1　大数の(弱)法則　平均値 μ，分散 σ^2 を持つ確率変数 X が与えられている．X からとった標本値 $X_1, X_2, \cdots, X_n$ と，それらの標本平均 $M_n = (X_1 + X_2 + \cdots + X_n)/n$ を考える．このとき，どんな小さな値 ϵ が与えられても

$$\operatorname*{limit}_{n\to\infty} P(|M_n - \mu| \geq \epsilon) = 0$$

である．

証　明

1. $X_1, X_2, \cdots, X_n$ は同じ**母集団**(population)からとったものであるので，

$$E(X_1) = E(X_2) = \cdots = E(X_n) = \mu$$

2. 3-13 節の式(3)を使えば

$$E(M_n) = \frac{n\mu}{n} = \mu$$

3. $V\left(\frac{X}{n}\right) = \frac{V(X)}{n^2}$ であるから 3-13 節の式(6)を使えば：

$$V(M_n) = V\left(\frac{X_1 + X_2 + \cdots + X_n}{n}\right) = \frac{n\sigma^2}{n^2} = \frac{\sigma^2}{n}$$

4. 確率変数 M_n に対して Chebyshev の不等式を適用すれば：

$$P(|M_n - \mu| \geq \epsilon) \leq \frac{\sigma^2}{n\epsilon}$$

5. どんなに小さな ϵ が与えられても，n が無限に大きくなるにつれて，右辺は 0 に近づくので，定理が証明される．

3-15 Bernoulli 試行：二項，多項，幾何分布

Bernoulli 試行は，最も簡単な種類の実験のほとんどを表現したものである．というのは，成功(s)と失敗(f)という 2 つの結果だけが許されているからである．s の確率は p であり，失敗の確率は $q = 1 - p$ である．すなわち，Bernoulli 試行とはくせのある硬貨を投げることであると考えればよく，p は表が出る確率で，$1 - p$ は裏が出る確率である．

Bernoulli 試行は，基本的な二項確率分布関数の実験モデルであり，この関数は特別な場合として Poisson および正規分布を含んでいる．

二項確率関数は $B(k;\ n,\ p)$ と表わされ，これは次のように読む：1 回の試行での成功の確率が p であるような Bernoulli 試行を n 回行なったときに k 回成功する確率．Bernoulli 試行を n 回連続して行なった結果は，おのおのアルファベット $\{s,\ f\}$ からとった n 個の記号の系列として表わされるので，いくつかの標本点は，たとえば $n = 7$ の場合には，次のようにラベルが付けられる．

$$ssfssff \qquad ffsfsss$$

それぞれの試行は互いに独立であるので，k 回の成功(したがって $n - k$ 回の失敗)を表わす標本点の確率は，積の規則によって次のようになる：

$$p^k(1-p)^{n-k}$$

これらの標本点の個数を求めるには，おのおのの標本点は，k 個の 1 を含む長さ n のビット・ベクトルであると考えればよい．そのようなビット・ベクトルの個数は $C(n,\ k)$ であることが第 3-1 節でわかっているので：

$$B(k;\ n,\ p) = C(n,\ k)p^k(1-p)^{n-k} \tag{1}$$

または

$$B(k;\ n,\ p) = \left(\frac{n!}{k!(n-k)!}\right)p^k(1-p)^{n-k} \tag{2}$$

ここで，$B(k;\ n,\ p)$は n 回の試行において k 回成功する確率であり，p は 1

回の試行における成功の確率である．通常，n と p は一定の値であり，一方 k は確率変数である．

例：　くせのないサイコロを5回ふった時に，2が

（a）　3回出る確率はいくらか．

（b）　少なくとも3回出る確率はいくらか．

解：　サイコロはくせがないので，どの目についても $p = 1/6$ である．(a)の解はしたがって $P(3;\ 5,\ 1/6) = (1/6)^3(5/6)^2(5!/2!\,3!)$ である．(b)の解は，"少なくとも3回の成功" という事象の標本点の確率を総計することによって求まり；

$$P(\text{少なくとも3回の成功}) = \sum_{k=3}^{5} B\left(k;\ 5,\ \frac{1}{6}\right)$$

$k = 0$ から $k = n$ までの B の値の総和は，もちろん1に等しくなければならない．これは，式(1)が

$$(p + (1 - p))^n = 1^n = 1$$

の二項展開の第 k 番目の項であることに気づけば，簡単に証明される．B の値の総和はこの二項式であり，したがってまた1に等しい．

二項分布の平均値を導出することは，次の定義から出発する．

$$E(k) = \sum_{k=0}^{n} kB(k;\ n,\ p) = \sum_{k=1}^{n} kB(k;\ n,\ p) \qquad (3)$$

これは，$k = 0$ に対する項が0だからである．

$$E(k) = \sum_{k=1}^{n} \frac{n!}{(k-1)!(n-k)!} p^k(1-p)^{n-k}$$

$k = j + 1$ と置きかえ，少々の代数を施せば

$$E(k) = pn\sum_{j=0}^{n-1} \frac{(n-1)!}{j!(n-1-j)!} p^j(1-p)^{n-1-j}$$

$n - 1 = r$ という別の置きかえを行なえば

$$E(k) = pn\sum_{j=0}^{r} \frac{r!}{j!(r-j)!} p^j(1-p)^{r-j}$$

この式の総和は**すべての** B の総和であるから1になり，したがって

$$E(k) = pn \tag{4}$$

これよりも多少退屈な手順により，k の分散に対する式が求まる．

$$V(k) = pn(1 - p) \tag{5}$$

二項分布は**多項分布**(multinomial)に一般化される．これは**おのおの**の試行には n 個の可能な結果が存在し，その i 番目のものの確率は p_i であるような実験に適用される．したがって

$$\sum_{i=1}^{n} p_i = 1 \tag{6}$$

このような試行を m 回連続したいかなる系列も，n のアルファベットに属する記号を m 個ならべた系列として結果を表わすことができる**実験**である．多項分布は，そのような系列が，結果のアルファベットの最初の要素を k_1 個，2番目の要素を k_2 個，$\cdots$，含んでいる確率を与えている．この確率は，二項式の場合と同じような論法を使って導出される．

$$P(k_1, k_2, \cdots, k_n; p_1, p_2, \cdots, p_n) = \frac{m!}{k_1!k_2!\cdots k_n!} {p_1}^{k_1}{p_2}^{k_2}\cdots{p_n}^{k_n} \tag{7}$$

ここで明らかに：

$$\sum k_i = k_1 + k_2 + \cdots + k_n = m \tag{8}$$

i 番目の結果が出現する回数の期待値(平均値)は

$$E(k_i) = np_i \tag{9}$$

であり，この結果に対する分散は：

$$V(k_i) = m(1 - p_i) \tag{10}$$

多項分布は，ある確率過程から，一度取り出した標本を元に戻す操作を許して標本をとったときに生じる度数分布を解析するのに役に立つ．度数分布は，可能な結果のおのおのが出現する回数を与えてくれる．おのおのの結果が相異なる記号で表わされるならば，与えられたいかなる度数分布も，多項モデルに適合するような記号系列となる．この事実は χ^2 検定(補遺Ⅱを見よ)の基礎となっている．χ^2 検定は，観察された度数分布と，ある抽象的な確率モデルから生じてくるような度数分布とを比較するためのものである．

二項分布は，Bernoulli 試行の連続によって定義できる確率関数であるだけ

ではない．たとえば，**幾何分布**が導かれるような次の実験を考えよう．

成功が**はじめて**出現するまで Bernoulli 試行を繰り返すことを考える．k 回の試行が必要である確率は：

$$G(k;\ p) = (1-p)^{k-1}p \tag{11}$$

与えられた k に対して標本点はただ1つしかないので，この式はたやすく確かめられる；この標本点には，$k-1$ 個の記号 f(失敗)の後に1個の記号 s (成功)をくっつけたものでラベルが付けられるであろう．試行は互いに独立であるので，積の規則によって式(11)が求まる．標本空間には無限個の点が含まれていることに注意しよう．なぜなら，われわれは，**はじめて**成功するまでこの試行を繰り返さなければならないからである．したがって，

$$\sum_{k=1}^{\infty}(1-p)^{k-1}p = 1 \tag{12}$$

この総和が1に等しいということは，幾何級数(等比級数)の和に関する公式を使って代数的に立証される．

負二項分布(negative binomial distribution)は，r 回目の成功が出現する**まで** Bernoulli 試行を繰り返した結果として生じるものである(ここでは，必要な試行の回数 k が確率変数である)．したがって，標本点のラベルは k 個の記号を持つであろう；初めの方の $k-1$ 個には $r-1$ 個の s と $(k-r)$ 個の f が含まれ，k 番目の記号は s でなければならない．与えられた任意の k に対し，このような形のものに積の規則を適用すれば，標本点の確率 $p^r(1-p)^{k-r}$ が求まる．このような標本点の個数は，初めの $k-1$ 個の位置に $r-1$ 個の記号をわりあてる方法の数であり，したがって：

$$N(k;\ r,\ p) = C(k-1,\ r-1)p^r(1-p)^{k-r} \tag{13}$$

$$\sum_{k=1}^{\infty}N(k;\ r,\ p) = 1 \tag{14}$$

3-16　Poisson 分布

多くの物理モデルで生じる Poisson 分布関数は多くの方法で導出される．この節では，二項分布の特別な場合としてそれを求める方法を示そう．この関数の再帰形から出発するのが一番便利である．この再帰形は，閉じた形である 3-15 節の式(2)から，簡単な代数によって求められる．

$$B(0;\ n,\ p) = (1-p)^n \tag{1a}$$

$$B(k+1;\ n,\ p) = \left(\frac{p}{1-p}\right)\left(\frac{n-k}{k+1}\right)B(k;\ n,\ p) \tag{1b}$$

式(1a)は $k=0$ の場合である．両辺の対数をとると：

$$\ln B(0;\ n,\ p) = n\ln(1-p) \tag{2}$$

$\ln(1-p)$を，$p=0$ の近傍で級数に展開し代入すると：

$$\ln B(0;\ n,\ p) = n\left(-p-\frac{p^2}{2}-\frac{p^3}{3}\cdots\right)$$

ここで，次のことを仮定する．

$$p \ll 1 \tag{3a}$$

$$k \ll n \tag{3b}$$

$$\lambda = pn \tag{3c}$$

そうすると，p が小さいので，級数の初項以外は省略できて：

$$\ln B(0;\ n,\ p) \fallingdotseq -np = -\lambda$$

逆対数をとれば

$$B(0;\ n,\ p) = P(0;\ \lambda) = e^{-\lambda} \tag{4}$$

ここで，式(3)の近似を式(1b)の再帰形に適用すると：

$$B(k+1;\ n,\ p) = \left(\frac{\lambda}{k+1}\right)B(k;\ n,\ p) \tag{5}$$

$B(0;\ n,\ p)$に対して式(4)を使い，式(5)に次々と代入を行なえば，次のものが得られる．

$$B(1;\ n,\ p) = \left(\frac{\lambda}{1}\right)e^{-\lambda} \tag{6a}$$

$$B(2;\ n,\ p) = \left(\frac{\lambda^2}{1\cdot 2}\right)e^{-\lambda} \tag{6b}$$

$$\cdots\cdots\cdots\cdots$$

したがって，一般に任意の k に対して：

$$P(k;\ \lambda) = \left(\frac{\lambda^k}{k!}\right)e^{-\lambda} \tag{7}$$

これが，Poisson 分布関数であり，また再帰形で書くと次のようになる．

$$P(0;\ \lambda) = e^{-\lambda}$$
$$P(k+1;\ \lambda) = \left(\frac{\lambda}{k+1}\right)P(k;\ \lambda) \tag{8}$$

確率関数に関する限り：

$$\sum_{k=0}^{\infty} P(k;\ \lambda) = \sum_{k=0}^{\infty}\left(\frac{\lambda^k}{k!}\right)e^{-\lambda} = 1 \tag{9}$$

これは，$e^{-\lambda}$ を総和からくくり出せるので，代数的に求められるであろう；$e^{-\lambda}$ をくくり出すと総和は e^{λ} の級数展開であることがわかる．$e^{-\lambda}e^{\lambda} = 1$ である．

Poisson 分布の平均値と分散は，二項分布のための式に式(3)の近似を適用すれば求められる．したがって：

$$E(k) = pn = \lambda \tag{10}$$
$$V(k) = pn(1-p) \fallingdotseq pn = \lambda \tag{11}$$

このように，Poisson 分布は，その平均値と分散に特徴がある．両者はともに，パラメータ λ に等しい．

ここで，近似が意味している“物理的な”状況あるいはモデルの状況を調べることは興味深い．式(3)は，Poisson 分布があてはまるのは，p(1回の試行での成功の確率）が非常に小さいような Bernoulli 試行の長い繰返しによって特徴づけられる実験であるということを示唆している；長い繰返し (n が大きい)によって得られる期待値(平均値)$\lambda = pn$ は，p が非常に小さくても，適度の大きさになる．したがって，p, n はパラメータ λ に“融合される”のである．われわれが扱っているのが Poisson 過程であることがわかっておれば，分布の平均値を示す単一のパラメータ λ さえあれば，確率関数の全体を決定するのに十分である．

次のこともまた，注目に値する．

$P(0;\ \lambda) = e^{-\lambda} =$ 成功が1回もない確率

$1 - P(0;\ \lambda) = 1 - e^{-\lambda} =$ 少なくとも1回は成功する確率

後の方の形は，Poisson 過程においては，“少なくとも1回は成功する”という確率は指数になるということを示している．Poisson 分布の表が，付表 C-1 に与えられている．

例：　ある通信システムには，サービスを要求できるユーザが100人おり，彼らは統計学的には差異がない．ある与えられた時間区間において，任意の

特定なユーザが要求を出す確率は0.05である(他のユーザについても同じである). 次のものを求めよ:

(a) 要求回数の期待値
(b) 少なくとも1人が要求を出す確率
(c) 3人以上のユーザが要求を出す確率

解: おのおののユーザの“要求を出す/出さない”を, Bernoulli 試行と考える. そうすると:

$$p = 0.05 \qquad n = 100 \qquad pn = (0.05)(100) = 5$$

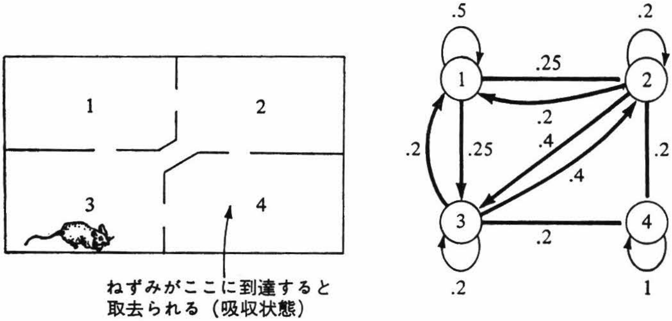
$p \ll 1$ であるから, このモデルは Poisson 過程と考えられる. したがって:

(a) $E(k) = \lambda = 5$
(b) $P(k \geq 1;\ 5) = 1 - P(0;\ 5) = 1 - e^{-5} = 1 - 0.0067$
$= 0.9933$
(c) $P(k \geq 3;\ 5) = 0.8753$(表から)

3-17 Markov 連鎖モデル

離散型の **Markov 連鎖**(Markov chain)モデルは, 概念的には, 計算機科学者にはなじみ深い有限状態オートマトン・モデルにいろいろな意味で似かよっている. **図 3-17.1** は基礎的な概念を説明するための例を示したものである. 部屋が4つある“ねずみの迷路”を考えよう. それぞれの部屋はシステムの状態を表わしている. 各部屋から他の部屋への出口はいろいろある. 各出口には, ねずみがその出口を通る確率が結びつけられている(確率が異なっているのは, 出口の穴の大きさが異なっているからである). 出口は, 状態を表わしている部屋を結びつけているので, 出口の確率は状態推移確率に相当する. また, この特定のモデルでは, ねずみは部屋4へ到達すると迷路から取去られるものと仮定する. 他のいかなる状態への推移確率も0であるこのような状態は, **吸収状態**(absorbing state)と呼ばれる.

最初, ねずみがある部屋(状態)に置かれるものと仮定しよう. ねずみが, 1, 2, 3, ･･･ステップの後で, ある決められた状態に居るであろうという確率はいくらであろうか. **ステップ**ということばは, 状態推移の意味である. このような質問に答えるために, 図 3-17.1 に示されている迷路問題の2つの表現をまず考えることから始めよう. 最初のものは状態図であり, そこでは, ねずみがある部屋に居るということが円で表わされた状態であり, 他の状態へ出て行く



(*a*) ねずみの迷路　　(*b*) 状態図

推移後の状態

現在の状態	1	2	3	4
1	.5	.25	.25	0
2	.2	.2	.4	.2
3	.2	.4	.2	.2
4	0	0	0	1

(*c*) 推移行列 P

$$
\underset{P}{\begin{matrix} .5 & .25 & .25 & 0 \\ .2 & .2 & .4 & .2 \\ .2 & .4 & .2 & .2 \\ 0 & 0 & 0 & 1 \end{matrix}}
\qquad
\underset{P^2}{\begin{matrix} .35 & .275 & .275 & .1 \\ .22 & .25 & .21 & .32 \\ .22 & .21 & .25 & .32 \\ 0 & 0 & 0 & 1 \end{matrix}}
\qquad
\underset{P^3}{\begin{matrix} .285 & .2525 & .2525 & .21 \\ .202 & .189 & .197 & .412 \\ .202 & .197 & .189 & .412 \\ 0 & 0 & 0 & 1 \end{matrix}}
$$

(*d*) それぞれ，丁度1,2,3ステップの後に任意の状態から別の状態に到達する際の確率行列

図 3-17.1　簡単な Markov 連鎖モデル

確率は，確率がラベル付けられた“有向線”で表わされている．“自己ループ”には，ねずみが同じ状態にとどまる確率がラベル付けられていることに注意しよう．また，問題の記述を行列の形で表わしたものも示されている．行列 P は**推移行列** (transition matrix) と呼ばれ，その P_{ij} 成分は状態 i から1回のステップで状態 j に推移する確率である．P の第 i 行を表わす P_i は，状態 i から出て行く線のすべてについて説明したものであり，したがって P_i のすべての成分の和は1のはずである．

それでは，前に提起された基本的な質問に戻って考えよう．ねずみが現在状態3に居るものとすれば，それが次のステップで状態1に居る確率はいくらであろうか．明らかにこの答は，状態図を見て状態3から状態1へ向かう矢印を捜し，そのラベルを読むことによって簡単に求められて，0.2であることがわかる．次に，以下の質問を受けたとする：状態3から出発し2回のステップの後で，ねずみが状態1に居る確率はいくらであろうか．これに答えるために，状態3から状態1へ向う2ステップの経路を注意深く追ってみよう．各経路に沿って，確率の積をとり（推移は独立であるから），それらの積の和をとると：

経 路	確 率
3 → 1 → 1	(0.2)(0.5) = 0.10
3 → 2 → 1	(0.4)(0.2) = 0.08
3 → 3 → 1	(0.2)(0.2) = 0.04
状態3から2回のステップで状態1へ推移する確率	= 0.22

推移行列からも(少々考察し実際にためしてみると)同じ結果が得られて:

$$\sum_{k=1}^{4} P_{3k}P_{k1}$$

あるいは，行列のベクトル表現を使えば，第3行と第1列の内積として表現できて:

$$P_{3,}\ P_{,1}$$

もっと一般的に言うと，行列 P とそれ自身の積は，次の意味を持つ．

$P^1 = P =$ 1回の推移の後での状態確率

$P^2 = PP^1 =$ 2回の推移の後での状態確率

また，

$P^n = PP^{n-1} = n$ 回の推移の後での状態確率

これから，Markov 連鎖の基本的な定義と定理をいくつかあげ，必要に応じて注釈をつけることにする．

定 義　正則 Markov 連鎖とは，その推移行列が，何回かべき乗されると，すべての要素が0でなくなるようなものを言う．

したがって，正則連鎖では，任意の状態からある有限回のステップを経ればすべての状態に到達することができるということになる．Markov 過程を通して進行して行くときには当然，P のべき乗の行列に関心があり，この成分は，前に見たように，任意の状態からおのおのの状態へ到達する確率を，経過したステップ数の関数として表わしている．次の定理は，正則連鎖では，到達しうる“定常状態”があるということを保証するものである．

定理 3-17.1　P が正則連鎖の推移行列であるとすれば，

1. n が増えるにつれて，P^n はある行列 T に近づく．
2. T のおのおのの行は，同一のベクトル W に等しい．
3. W のすべての要素は 0 ではない．

定理 3-17.2　**定理 3-17.1** の定常状態確率ベクトル W は，唯一であり，ベクトルと行列の方程式 $WP = W$ を満足する．

正則連鎖は，**エルゴード連鎖**(ergodic chain)の部分クラスである．

定　義　エルゴード連鎖とは，どんな状態にも，いくつかのステップを経過すれば，他のすべての状態から到達できるような Markov 連鎖のことである．

エルゴード・クラスは，正則クラスより広範なものである，すなわち，すべての正則連鎖はエルゴード的であるが，エルゴード連鎖のすべてが正則ではない．たとえば次のような，状態数が 2 つで，状態 1 から状態 2 へは（またその逆も），奇数回のステップでしか到達できないようなシステムの推移行列を考えよう．

$$P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad P^2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad P^3 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \text{など}$$

ここでは，すべての状態は，1 または 2 回のステップで，他のすべての状態から到達可能であり，したがって連鎖はエルゴード的である．しかしながら，P のべき乗は常に 0 要素を含んでいることがわかるので，連鎖は正則でない．これから，エルゴード連鎖に関する重要な定理をいくつか挙げよう．

定理 3-17.3　P がエルゴード連鎖の推移行列であれば，

1. $WP = W$ となるような唯一の確率ベクトル W が存在する．この W は，**定常状態確率ベクトル**と呼ばれる．
2. W のすべての要素は 0 でない．
3. 最初の n 回のステップの間に，過程が状態 s_j に存在する回数の平均値を $h_j^{(n)}$ とすると，任意の $\epsilon > 0$ に対し，

   確率$(|h_j^{(n)} - W_j| > \epsilon) \to 0$　　（どんな状態から出発しても）

この定理は，Markov 連鎖に対する“大数の法則”である．なんとならば，

この定理は，次のことを大ざっぱに述べたものであるからである：状態推移の回数が増大するにつれて，ある与えられた状態にシステムが存在する回数の平均値(すなわち，その状態に存在している相対度数)は，システムがその状態に存在する定常状態確率に近づく．

しばしば Markov 連鎖では，推移をくり返すうちに，ある状態を去っても何回かのステップの後にそこに戻ってくることがしばしばある．ある状態に戻るのに必要なステップ数を，その状態の**再帰時間**(recurrence time)と呼ぶ．

例：　あるジョブが，可能な状態は2つしかないものとして表わされているとしよう．最初のものはCPU 処理であり，二番目はI/O 処理である．すべての状態推移について確率が与えられているとしよう．その推移確率を使って，CPU と I/O の状態の定常状態確率を表わす式を求めよ．

解：　推移確率は

$$P = \begin{pmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{pmatrix}$$

P_{ij} はすべて0でないので，連鎖は正則かつエルゴード的である．W_1 と W_2 を定常状態の確率とすれば，**定理 3-17.3** から：

$$(W_1,\ W_2)\begin{pmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{pmatrix} = (W_1,\ W_2)$$

これから，次のはじめの2つの方程式が得られ，これに3番目の方程式を加えると：

$$\begin{aligned} P_{11}W_1 + P_{21}W_2 &= W_1 \\ P_{12}W_1 + P_{22}W_2 &= W_2 \\ W_1 + \quad W_2 &= 1 \end{aligned}$$

これは，2つの未知数を含む3つの方程式であるので，方程式は独立でない．最後の方程式が成りたた**なければならない**ので，後の2つの方程式を選ぶ．それら両方を解いて，次式を得る．

$$W_1 = \frac{1 - P_{22}}{1 + P_{21} - P_{22}}$$

$$W_2 = \frac{P_{21}}{1 + P_{21} - P_{22}}$$

定理 3-17.3 から，W_1, W_2 はそれぞれ，システムが CPU 状態と I/O 状

態に存在する回数の平均値になる．

定理 3-17.4　エルゴード・Markov 連鎖では，状態 i の平均再帰時間は $1/W_i$ である．ここに，W_i は定常状態確率ベクトルの i 番目の成分である．

吸収状態は，“ねずみの迷路”の例で説明したように他の状態への出口がない状態であることがわかった．推移行列の対応する行は，主対角成分にあたる部分が“1”である以外はすべて0である．過程が一度吸収状態に到達すると，それはブロックされてしまう(すなわち，それ以上別の状態に到達することができない)．連鎖が吸収状態を含んでおり，その状態に到達可能であれば，連鎖は吸収していると言われる．1つ以上の吸収状態を持った Markov 連鎖は，いつでも，その**推移行列の標準形**(standard or canonical form)で表わすことができる．

$$P = \begin{array}{c} \\ r \\ s \end{array}\!\!\begin{array}{c} \begin{array}{cc} r\text{ 状態} & s\text{ 状態} \end{array} \\ \left(\begin{array}{c|c} I & O \\ \hline R & Q \end{array}\right) \end{array}$$

ここでは，最初の r 個の状態は吸収状態である．残りの s 個の状態はそうでない．このとき

$I = r \times r$ の単位行列(主対角成分は1でそれ以外は0)
$O = r \times s$ の0行列
$R = s \times r$ の行列．非吸収状態から吸収状態への推移行列
$Q = s \times s$ の行列．非吸収状態から非吸収状態への推移行列

P の n 回のべき乗は次のように書かれるということが簡単にわかる．

$$P^n = \begin{pmatrix} I & O \\ * & Q^n \end{pmatrix}$$

ここに，$*$ は，直接には興味がない行列を表わしている．P と P^n が似ていることに注意しよう．特に，Q^n は，任意の非吸収状態から出発して n 回のステップの後に非吸収状態に存在している確率を示している．このことを心に留めて，次に2つの重要な定理を掲げる．

定理 3-17.5　吸収型の Markov 連鎖では，到達しようとしている吸収状態の確率は1である．

定理 3-17.6　N は，次に示すように，P の標準形の部分行列に関係した行

列とする．

$$N = (I - Q)^{-1}$$

このとき，N の成分は，可能な非吸収状態のおのおのから出発した過程が各非吸収状態に到達する回数の平均値を与えている．

3-18　信頼性モデル：直列-並列構造

システムの信頼性の考察は，次のような見地からなされる．

1. 故障(または信頼度)の見積り
2. 故障の確率の最小化
3. 故障の発見
4. 故障からの回復

この章でわれわれが取り扱うのは，これらのうちの1番目の項目，もっと正確には，システムの信頼度をその構成要素の信頼度から計算することである．後で展開される数学的モデルは，さまざまな形態のシステムの信頼性の性能に対してある洞察を得るのに貴重なものとなる．しかしながら，構成要素(磁気テープ，ディスク，記憶装置，CPU 等)の信頼度のデータを手に入れることはほとんどできず，明らかにこのことによって，どんな予測手法も実用的価値に限界がある．

信頼性は，電子式機械の昔から現在に至るまで，計算機設計では不可欠の考察事項である．最近のシステム設計では，装置を用いたシステムの製作だけでなくプログラミングの信頼性も考察する．その例としては，最も簡単でかつ最も信頼性が高い2-状態で動作する機器を選ぶこと，回路は最悪の場合を考慮して設計すること，I/O や記憶装置やデータの経路におけるパリティチェック，そして，いくつかの機械的な機器で行なわれる**エコーチェック**(echoing)や自動再実行などのもっと手の込んだ方法などがある．(**エコーチェック**は，機器に入力された情報のチェックを，機器側でコピーして返送してきた情報と元の情報を比較することによって行なう手法である．)

欲しい品物を手に入れるには金を払わなければならないのと同じように，信頼性は費用がかかるものである；最も信頼性の高いシステムでも，費用がかかりすぎれば，実用上は最良のシステムとは言えない．国家の安全，人間の生命，会社の最重要な業務などが決定的な場面でシステムに依存している場合には，当然，信頼性に投資することは，不可欠とは言わないにしても，賢明な策であ

る；信頼性を改善するにはどれ位投資すればよいかを決めるときの1つの因子は，そのシステムのユーザの立場から見て評価された故障コストである．

一般的なシステムの信頼性を扱う学問は，最近では技術的方法論としてすっかり1人前となっており，このことは，この分野で仕事をする人のための教科書，数多くの技術論文，専門教育などで見られるとおりである．信頼性の理論は発展しつつあるが，組み合せ数学，確率論，統計学などがこの発展に大いにあずかっている．たいていの理論でもそうであるように，数学的な手法は，現実世界を理想化したあるモデルに適用される．この章の残りの部分では，そのようなモデルのいくつかを論ずる．

どんな時点においても，稼動しているか稼動していない(故障している)かが分類できるような，よく定義されたシステムの部分はどれでも，**構成要素**(component)と呼ばれる．構成要素の**信頼度**は，ある決められた時間間隔にわたってそれが稼動するであろう(故障しないであろう)という確率として定義される．r が構成要素の信頼度であれば，$1-r$ はその構成要素の故障の確率[しばしば**非信頼度**(unreliability)と呼ばれる]である．システムは，構成要素を相互に結合したものから成っている．技術的な見地からすれば，システムや部分システムもまた構成要素であることに注意しよう．以後われわれは，すべての構成要素の故障は独立であると仮定する．これから行なう議論では，われわれは物理的な相互結合には直接関心がなく，信頼性構造に関心がある．しばしばこの構造は，信頼性の見地から見たシステムのふるまいを表わすネットワーク様の図で表現することができる．いずれも n 個の構成要素を含んだ，2つの基本的な信頼性構造が対比される．その最初のものは，**直列**システムと呼ばれ，どれか1つの構成要素が故障すればシステムが故障してしまうという特性を持っている．別の言い方をすれば，直列構造とは，システムが稼動するためにはその構成要素のすべてが稼動しなければならないように結合された構造である．直列構造の信頼度は，その構成要素の信頼度の**積**である．一般的に言うと，n 個の構成要素を直列に含み，その i 番目の構成要素の信頼度が r_i であるような信頼性構造では，システムの信頼度 R は次のようになる．

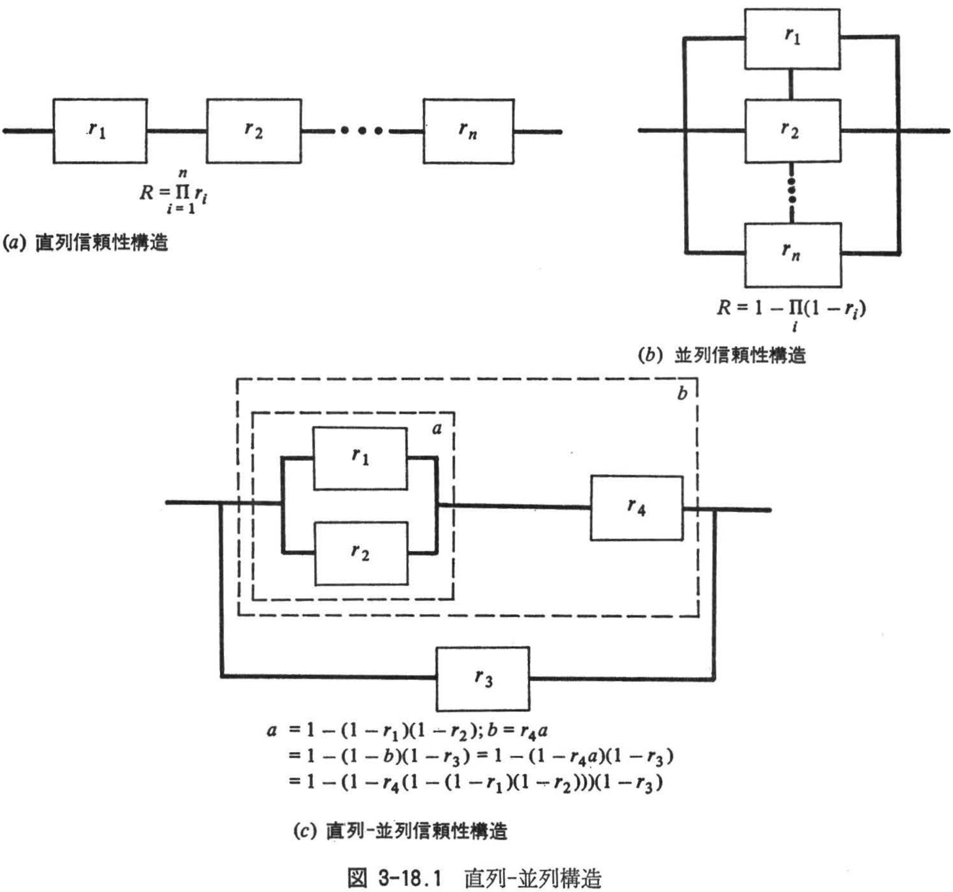
$$R=\prod_{i=1}^{n} r_i \tag{1}$$

もう1つの簡単な信頼性構造は**並列**と呼ばれ，すべての構成要素が故障したときに限りシステムが故障するという特性を持っている．この場合には，システムの故障確率は，構成要素の非信頼度の積になる．システムの信頼度は1か

ら故障確率を減じたものであるから：

$$R = 1 - \prod_{i=1}^{n} (1 - r_i) \qquad (2)$$

多くのシステムの信頼性構造は，直列および並列構造の組み合わせで構成されていると考えることができる．そのような部分構造に上記の基本的な関係式が次々と適用されて，最後に構成要素の信頼度を使って表わされたシステムの信頼度の式が求められる．この過程を1歩1歩行なった例が**図 3-18.1** に示されている．r_1, r_2 から成る並列な部分構造の信頼度は a で表わされ，これを表



(*a*) 直列信頼性構造

(*b*) 並列信頼性構造

(*c*) 直列-並列信頼性構造

図 3-18.1　直列-並列構造

わした式と r_4 を組みあわせて，直列な部分構造の信頼度 b を求める．最後に，b と r_3 を並列構造として組みあわせ，p を表わす式を求める．

別の例として，構成要素の信頼度が次のようになっている計算機システムを考えよう．

信頼度が r_1 の磁心記憶装置が1台
信頼度が r_2 の演算装置が1台
信頼度が r_3 のディスク装置が2台
信頼度が r_4 の磁気テープ装置が3台

このシステムが稼動するためには，磁心記憶装置，演算装置，少なくとも1台のディスク装置，少なくとも2台の磁気テープ装置が稼動しなければならない．このシステムの信頼度はいくらであろうか．明らかに，この信頼性の構造は，r_1 と r_2 とを直列に結び，この直列構造と2つの並列な r_3 とを直列に結んでできる部分構造を含む．この部分構造は，r_4 を含んだ部分構造と直列に結ばれる．"3台のうちの2台" のテープという仕様に対する信頼度を表わす式を求めるには，3台のテープを表わす8行の真理値表を作ればよい．1を少くとも2つ含む行に対応する関数欄には "1" を置く．これらの行とそれに対応する信頼度の式は次のようになる．

$$
\begin{aligned}
011 &\to (1-r_4)r_4r_4 = r_4^2 - r_4^3 \\
101 &\to r_4(1-r_4)r_4 = r_4^2 - r_4^3 \\
110 &\to r_4r_4(1-r_4) = r_4^2 - r_4^3 \\
111 &\to r_4r_4r_4 = r_4^3
\end{aligned}
\tag{3}
$$

システムの磁気テープ部分の信頼度は，上の式の総和で：

$$t = 3r_4^2 - 2r_4^3 \tag{4}$$

ついで，この信頼度と，前に述べたシステムの残りの部分とを直列に組みあわせる．したがって，システム全体の信頼度の式は次のようになる．

$$R = r_1r_2(1-(1-r_3)^2)r_4^2(3-2r_4) \tag{5}$$

3-19　より一般的な信頼性構造

多くの信頼性に関する記述は，上に見たような直列-並列構造に簡単に翻訳することができる．これが不可能なタイプの1つは，**3-18節**の後の例に見られたような "少なくとも，m 個のうちの j 個の構成要素が稼動しなければならない" というタイプの記述である．その場合にわれわれがたよりとするのは，

真理値表の手法であり，これは次のように一般化される：おのおのの信頼度が r である m 個の構成要素が与えられている．システムが稼動するためには少なくとも j 個の構成要素が稼動しなければならないとすれば，信頼度は次のようになる．

$$R = \sum_{i=j}^{m} C(m, i) r^i (1-r)^{m-i}$$

この公式は，真理値表による表現法，積の規則，**3-1 節**で見た $C(n, i)$ についての議論などを用いて導かれる．

3-20　冗　長　性

今までは，信頼性の解析問題，すなわち，与えられたシステムの信頼度の計算のみを考察してきた．指定の構成要素によって信頼性を改善する問題をこれから調べていこう．

あるシステムが n 個の構成要素により規定されており，そのシステムの信頼度が p であるとき，その信頼度を改善するためには，通常，構成要素を付け加える必要があるだろう．そのとき，システムには**冗長性**がある．実際には，適切な設計により，付加される構成要素は性能を改善するためにも使われてもよい．そのようなシステムでは，すべての構成要素が稼動するとき，最大の性能(たとえばスピード)が得られる．故障が生じ，故障した構成要素の数が少なくともある限度までなら，システムは性能の低下をこうむるが，しかし依然として動作可能である．このようなシステムは，“**ソフトに故障する**(fail softly)” と言われる．

さて，与えられた量の冗長な機器を最もうまく使うにはどうすればよいかという基本的な質問について考えよう．システムがもともと，それぞれの信頼度が r の直列な構成要素を s 個必要とし，そして，われわれは ms 個の構成要素を使うこともいとわないとすれば，2つの対照的な方式が存在する．それらは，**構成要素待機**(component standby)および**システム待機**(system standby)と呼ばれる．構成要素待機では，おのおのの構成要素は $m-1$ 個の構成要素と並列に並べられ，そしてこの並列構造が s 個直列に結ばれる．したがってシステムの信頼度は

$$R = (1-(1-r)^m)^s \qquad (1)$$

システム待機では m 個のシステムが並列になっており，そのおのおのは，直列な s 個のコンポーネントを含んだ元のシステムを複製したものである．シス

テムの信頼度は

$$R = 1 - (1 - r^s)^m \tag{2}$$

任意の信頼度 r と与えられた s と m の値(機器の量)に対して，構成要素待機方式は少なくともシステム待機方式と同じくらいの信頼度があるということを示すことができる．しかし，構成要素待機システムの方が結線の数が多くなり，したがってよりコストがかかる．また，切り換え操作の信頼性も考慮しなければならない．

3-21　故障までの平均時間

今まではずっと，構成要素の信頼度を記号 r で表わしてきた．実際には，r は時間の関数になるであろう．理論的な結果を導出したり，計算を行なったりするのに特別に適した理論的モデルがいくつか存在する．しかし，言うまでもないことであるが，これらのモデルから得られる結果を実際の場面に適用するときには注意が必要である．

$r(t)$ に対する一般的な解析式を議論する前に，**図 3-21.1** に示すような，より実際的な場合を定性的に考えるのがよいであろう．この曲線の形は，真空管や機械的な構成要素の多くに典型的なものである．この曲線は３つの領域を持っている．最初のものは**初期故障期**あるいは**幼児期**と呼ばれ，導入された時点から時刻 t_1 まで続く．その後，故障率が一定の期間が t_2 まで続く．そして最後に，**摩耗故障期**がある．この関数は，現実のシステムの多くをかなりの程度代表するものではあるけれども，解析的にこれを表現することは困難である．以下に述べる理論は，故障率が一定の平坦な領域にあてはまるものである．

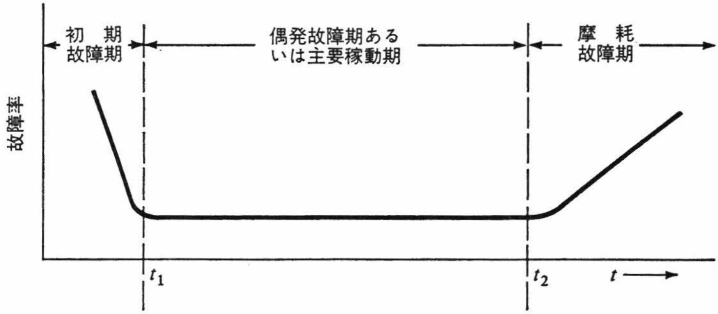


図 3-21.1　時間に対する故障率の典型的な曲線

長い時間にわたって，数多く(n_0 個)の同一の構成要素を観察することを考える．時間がたつにつれて，いくつかの構成要素が故障するであろうから，生き残りの数 n_s は時間の関数となるだろう．時刻 t における構成要素の信頼度は，その時刻に生き残っている数と元の数との比として定義される．

$$r(t) = \frac{n_s(t)}{n_0} \tag{1a}$$

構成要素は次のようにして故障するものと仮定する．すなわち，t から $t+dt$ までの時間区間では，故障する数 dn_f は区間のはじめで生き残っている数と区間の長さとの両方に比例するものとする．

$$dn_f = an_s dt \tag{1b}$$

$n_f = n_0 - n_s$ であり，また n_0 は t に無関係であるから，式(1b)は次のようになる．

$$-dn_s = an_s dt \tag{1c}$$

式(1a)から，$n_s = n_0 r$ であるので，式(1c)は次のように書かれる．

$$\frac{dr}{r} = -a dt \tag{1d}$$

この両辺を積分し，条件 $r(0) = 1$ によって積分定数を決めると，指数の法則が得られる．

$$r(t) = e^{-at} \tag{1e}$$

信頼度と故障の密度関数は，r と $1-r$ の導関数である．

$$r' = \frac{dr}{dt} = -ae^{-at} \tag{2a}$$

$$(1-r)' = ae^{-at} \tag{2b}$$

故障までの平均時間(Mean Time to Failure: 略して **MTF**)は，故障密度の全時間についての平均値である．

$$\mathrm{MTF} = \int_0^\infty t(1-r)' dt \tag{3}$$

$(1-r)'$ を求めるのに式(2b)を使えば:

$$\mathrm{MTF} = \int_0^\infty tae^{-at}dt \tag{4}$$

部分積分によって

$$\mathrm{MTF} = \frac{1}{a} \tag{5}$$

同一の構成要素が n 個**直列に**並んだものに対しては，構成要素の信頼度の指数分布によって次の式が求まる．

$$R(t) = [r(t)]^n = e^{-nat} \tag{6}$$

このシステムの故障までの平均時間は，したがって：

$$\mathrm{MTF} = \frac{1}{na} \tag{7}$$

式(5)と比べれば，これは，単一の構成要素の故障までの平均時間の $1/n$ 倍になっていることがわかる．

より一般的に，信頼度が同じでない n 個の構成要素が直列に並んでおり，その i 番目の構成要素が信頼度 r_i および MTF $t_i = 1 \div a_i$ を持つとすると，式(6)は積となり，指数分布に対しては，a_i の総和が指数の部分に現われる．このようなより一般的な場合には，式(7)は次のようになる．

$$\mathrm{MTF} = 1 \div \sum_{i=1}^{n} a_i = 1 \div \sum_{i=1}^{n} (1 \div t_i) \tag{8}$$

同一の構成要素から成る並列システムの故障までの平均時間は，代数的な取扱いはより複雑になるけれども，式(7)を求めたのと似た過程によって導出することができる．式(3)の定義に，3-18 節の式(2)を代入すると：

$$\mathrm{MTF} = \frac{n}{a}\int_0^\infty at(1 - e^{-at})^{n-1}e^{-at}d(at) \tag{9}$$

$1 - e^{-at}$ の $(n-1)$ 次のべき乗に対して二項級数展開を行ない，次いで積分と簡単化を行なえば次のものが求まる．

$$\mathrm{MTF} = \frac{1}{a}\sum_{j=1}^{n} (-1)^{j-1}\frac{C(n, j)}{j} \tag{10}$$

ここに，$C(n, j)$は二項係数(n 個のものから一度に j 個をとる組み合せの数)である．式(10)は次のように簡単になることがわかる．

$$\mathrm{MTF} = \frac{1}{a}\sum_{j=1}^{n}\frac{1}{j} \tag{11}$$

3-22　故障/修復 信頼性モデル

今まで考えてきたモデルはすべて，故障の確率のみを扱うものであった．実際には，機械(信頼性の構成要素)に故障が生じると，故障診断を含む修復が行なわれる．われわれは今から，故障と修復の両方を含んだモデルを考えることにする．

待ち行列理論の文献で**機械干渉**(machine interference)**モデル**と呼ばれているものは，Markov モデルである．第5章で見られるように，これと同じ数学的手法が，いろいろなタイプのマルチユーザ・システムに適用される．このモデルを信頼性の分野の言い方で言うと，全部で n 個の機械があり，それらのうち $n-i$ 個は稼動しており，i 個は“ダウン”している，すなわち，稼動しておらず修復中である．稼動中の機械が故障する確率はどれについても同じで，また故障中の機械の修復が完了する確率もどれについても同じであると仮定する．さらに，おのおのの機械の，故障と修復の確率は定数で，故障した時刻や，過去の履歴に無関係であると仮定する．また，故障や修復完了はすべてお互いに独立であると仮定する．

このモデルは，修復施設，すなわち窓口があり，各故障がそこでのサービスを要求して待ち行列を作っている状況であると考えることができる．そこで，

n = 機械の総数

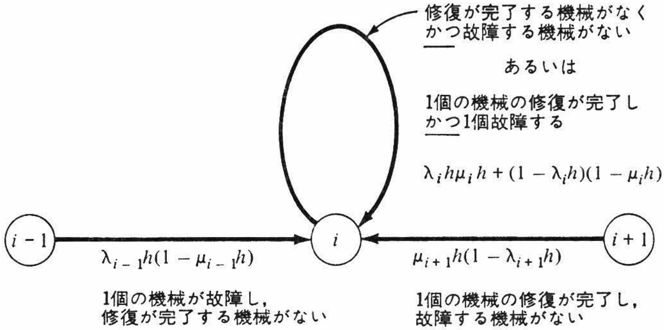
P_i = i 個の機械が修復中である(稼動していない)確率

$\lambda_i h$ = 短い時間区間 h の間に，ある1つの稼動中の機械が故障する確率

$\mu_i h$ = 短い時間区間 h の間に，ある1つの故障中の機械の修復が完了する確率

とする．それぞれ故障率，修復率を表わしている，λ_i と μ_i とは，現在故障中の機械の個数(i)の関数であることに注意しよう。われわれは後で，λ_i と μ_i に関する特定な関数を提起するが，今のところは，一般的な解析を続ける．

さて，**図 3-22.1** に示される，Markov 型の表現を考えよう．状態 i は，i 個の機械が修復を待っており，当然，$n-i$ 個の機械が稼動中であることを意味している．状態推移は時間区間 h でのみ生じる．2個以上の機械がその時間区間内で故障したり修復が終ったりする確率が無視できるように，h は十分短かくなければならない．後で簡単に触れるように，h はこの問題のパラメータではなく，図 3-22.1 に示されるように，非常に小さいと仮定されなければばな



$\lambda_i h$ = i 個の機械が故障中にどれか1個が故障する確率
$\mu_i h$ = i 個の機械が故障中にどれか1個の修復が完了する確率

図 3-22.1　故障-修復モデルに対する Markov ノード

らない．というのは，$i-1$ および $i+1$ から i へという1状態の推移のみを含む状態推移を考えるからである．図3-22.1から，状態 i の確率は，ノード i へ流入する線についての総和をとることによって求まる．求まった結果で，h^2 を含む項を無視すれば(なんとなれば，前に述べたように h は小さいと仮定されているから)，次のようになる．

$$P_i = P_{i-1}\lambda_{i-1}h + P_i(1-\lambda_i h - \mu_i h) + P_{i+1}\mu_{i+1}h \qquad (1)$$

両辺から共通な P_i を引き，h で割り，P_{i+1} について解けば：

$$P_{i+1} = \frac{1}{\mu_{i+1}}(P_i(\lambda_i + \mu_i) - P_{i-1}\lambda_{i-1}) \qquad (2)$$

どの機械も故障していないときには，修復が完了する機械はないので：

$$\mu_0 = 0 \qquad (3)$$

また，n 個の機械がすべて修復中であれば，故障するものはありえないので：

$$\lambda_n = 0 \qquad (4)$$

$i=0$ に対しこれを使えば，式(2)から次のようになる．

$$P_1 = \frac{\lambda_0}{\mu_1}P_0 \qquad (5)$$

式(2)を次々と使っていけば，まず

$$P_2 = P_0 \frac{\lambda_0 \lambda_1}{\mu_1 \mu_2} \tag{6}$$

が得られ，一般形は次のようになる．

$$P_j = P_0 \frac{\lambda_0}{\mu_j} \prod_{i=1}^{j-1} \left(\frac{\lambda_i}{\mu_i} \right) \qquad 1 \leq j \leq n \tag{7}$$

P_j は，j 個の機械が修復中であるという確率であるから，これはまた，$n-j$ 個の機械が稼動中であるという確率でもある．したがって，稼動中の個数の平均値は

$$\overline{n} = \sum_{j=0}^{n} (n-j) P_j = n - \sum_{j=1}^{n} j P_j \tag{8}$$

となる．ここでは，

$$\sum_{j=0}^{n} P_j = P_0 + \sum_{j=1}^{n} P_j = 1 \tag{9}$$

の関係を使った．式(7)を式(9)に代入して P_0 について解けば：

$$P_0 = \frac{1}{1 + \sum_{j=1}^{n} \frac{\lambda_0}{\mu_j} \prod_{i=1}^{j-1} \left(\frac{\lambda_i}{\mu_i} \right)} \tag{10}$$

さらに，特定な結果を得るためには，λ_i, μ_i を i の関数として明確に表わすことが必要になる．“理にかなった”そのような関数をいくつか提案することができる．それぞれは，故障/修復機構の異なる表現に対応し，これらの表現は大ざっぱなものではあるが，利用できる見込みがある．たとえば，機械の故障の確率はみな同一で互いに独立であるから，$n-i$ 個の機械が稼動中に，ある1つの機械が故障する確率は次のようになると考えられる．

$$\lambda_i = (n-i)\lambda \tag{11}$$

ここで，λ は1つの機械の故障率である．同じように，i 個の機械が修復中で，修復が完了するのはそのうちのどれでもよいと仮定すれば：

$$\mu_i = i\mu \tag{12}$$

この最後の仮定は，非常にのみこみにくいものであるが，どうしてもこうするのである！　システムの信頼度 R は，修理中でない機械が少くとも1台はあるという確率として定義される．すなわち，

$$R = 1 - P_n$$

P_n に対して式(2)および式(11)，(12)を使い，簡単化を行なうと次のようになる．

$$R = 1 - \frac{1}{1 + \sum_{j=1}^{n} \left(\frac{\mu}{\lambda}\right)^j C(n, j)} \tag{13}$$

ここに

$$C(n, j) = \frac{n!}{j!(n-j)!}$$

である．**図 3-22.2** は，いろいろな値の比 μ/λ を持つ5個までの機械についての R を示したものである．

本節と前の節では，信頼性の理論を紹介してきた．数学的モデルには，いくつかの仮定が暗黙に含まれていた．その主なものの1つは，おのおのの構成要素の故障の性質は他のすべてのものと独立であるということである．また，予防的な保全や，その他のタイプの保全については無視してきた．

時間に依存する指数の信頼度を仮定することは，理論的な研究において重要である，というのは，これは数学的な取扱いに便利であり，特に，信頼性の改善の限界を，構成要素の構成形態の関数として求めるときに便利だからである．**指数の法則**は，次の仮定と同値である：与えられた任意の時間区間の信頼度は，その区間の出発時点における構成要素の年齢に無関係である．信頼性に関する文献では，指数以外の分布も取扱われている．

von Neumann, Moore, Schannon らの初期の仕事で，信頼性の低いコンポーネントから信頼性の高い構造を作りあげるための理論的な方法がいくつか

	$\frac{\mu}{\lambda}$ = 故障までの時間/修復完了までの時間					
N	0.1	0.5	1	2	5	10
1	0.091	0.333	0.500	0.667	0.833	0.909
2	0.174	0.556	0.750	0.889	0.972	0.992
3	0.249	0.704	0.875	0.962	0.995	0.999+
4	0.317	0.802	0.938	0.988	0.999+	0.999+
5	0.379	0.868	0.969	0.996	0.999+	0.999+

図 3-22.2　機械の個数と故障-修復時間をいろいろ変えた場合のシステムの信頼度

確立された．

信頼性の理論は，情報理論，特に符号化理論に関係がある．これら2つの学問は互いに生産的な影響を及ぼしあっていくだろうという強い証拠がある．

演　習　問　題

1． ほとんどの計算機は，その電気回路では内部記号0と1を使う．

(a) 1バイトは8ビットから成り，各バイトは文字セットの1文字を表わすならば，可能な文字セットの大きさはいくらか？

(b) ある計算機のアドレス・レジスタは24ビットの大きさである．アドレス可能なセルは何個あるか？ (10進数で表わせ)

(c) ある計算機は56個の相異なる文字から成る文字セットを持つとする．1文字あたりに何ビット必要か？

2． 7つのジョブの集合から次のような条件で4つのジョブのストリームを選び出す方法は何通りあるか？

(a) 同じジッブが異なる順に並んだストリームは，別のストリームとして数えられる．

(b) 順序が異なっても別のストリームとしては数えない．

3． ある計算機システムの文字セットには48個の文字があり，80列12行の標準IBMカードが使われる．

(a) この文字セットを表わすためには，カードの1列あたりに最大何個の穴が必要か？

(b) カードには80列あり，1列で1文字を表わすとすれば，この48個の文字セットを使ってカード1枚で何種類の文字系列を表わすことができるか？ (計算して結果を出す必要はなくその数学的表現を示すだけでよい．)

4． 通常のIBMパンチ・カードは1列あたりに12個の穿孔位置がある．普通には，1列内の穴の形状はデータの文字を表わす．最大h個($1 \leq h \leq 12$)の穴を許した際に得られる可能な文字数(すなわち，文字セットの大きさ)$N(h)$を並べた，12行の表を作れ．

5． ある計算機メーカーが作っている装置には，次のようなモデル系列がある．

CPU	3モデル

主記憶装置	6サイズ
ディスク駆動装置	4モデル
磁気テープ駆動装置	4モデル
カード・リーダ・パンチャ(CRP)	1モデル
プリンタ	3モデル

次の場合に，相異なるシステム構成は何種類考えうるか？

(a) CPU 1台，主記憶装置1台，ディスク駆動装置2台，磁気テープ駆動装置2台，CRP 1台，プリンタ1台.

(b) CPU 1台，主記憶装置2台，ディスク駆動装置を少くとも2台，磁気テープ駆動装置を少くとも1台，CRP 1台，プリンタを2台まで.

6. 10人の技術者，15人のプログラマ，12人のセールスマンから成る候補者集団の中から，3人の技術者，4人のプログラマ，2人のセールスマンから成る9人の機動部隊を構成する方法は何通り考えうるか？

7. ある会社では，データ処理に使うために在庫品目録内の各項目に相異なる部品番号を割当てる必要がある．80000個の相異なる部品が在庫している．

(a) アルファベット2文字と数字2文字をこの順に並べた番号で，相異なる番号を必要な個数作れるか？

(b) 部品番号は，アルファベット2文字と数字2文字を任意の順に並べたもので表わされるとする．何個の部品番号を表わすことができるか？

8. いくつかのオペレーティング・システムは主記憶装置をパーティションに分割し，ジョブ・スケジューリングの1方法としてジョブをパーティションに割当てる．主記憶装置が5つのパーティション(P)に分割されるなら，主記憶は次のようになるであろう．

$$|P_1|P_2|P_3|P_4|P_5|$$

各パーティションには，異なるクラスのジョブをスケジューリングするためにクラス記号 A または B または C が割当てられる．

(a) 各パーティションには繰返しを許して1つのクラス記号が割当てられるならば，相異なるクラス割当ては何通りできるか？

(b) 各パーティションには1つのクラス記号が割当てられ，どのクラ

ス割当てにおいてもすべてのクラス(A, B, C)が出現しなければならないとすれば，相異なるクラス割当ては何通りできるか？

9. ある計算機は，独立にアクセス可能な32個のモジュールから成る主記憶装置を持っている．この主記憶装置に対するアドレス参照はどのモジュールに対しても同じように起こる．主記憶制御部は十分多くの要求のストリームからモジュールへの参照を行なうと仮定する．

(a) 要求が2個のストリームに対し，相異なる参照の方法は何通り考えうるか？

(b) 上の問題 (a) で要求が20個とした場合に考えうる数多くの参照の方法のうち，第1番目のメモリ・サイクルの間に相異なる8個のモジュールへの参照が許されるものは何通りあるか？

(c) 要求が20個の参照ストリームのうちで，20個の要求を処理するのに必要なメモリ・サイクルのうちの1つのメモリ・サイクルの間に相異なる8個のモジュールへの参照が許される参照方法は何通りあるか？　これは非常にむずかしい問題であることに注意せよ．これは読者に頭をひねってもらうために取り入れられた問題である．またこれは，問題文を少し変えるだけで問題の複雑さがどれくらい変り得るかを示す実例である．

10. ある計算機センターには100人のユーザがいてそのうちの50人はFORTRANあるいはCOBOL(または両方)を使い，20人はFORTRANだけを使う．もし可能ならば与えられた情報から次の問に答えよ．上記が与えられただけでは答えることができないならば，付け加えるべき情報を示せ．

(a) FORTRANもCOBOLも使わない人数

(b) COBOLを使う人数

(c) COBOLだけを使う人数

11. あるプログラミング作業は4つのモジュール A, B, C, D に分割された．動員されるプログラマはチームに入ってモジュールの作業を行なうことになっており，各チームには1つまたは複数個のモジュールが割当てられる．

(a) プログラミング作業の割当て方法は何通り考えられるか？（注意：各方法についてチームの数は同じである必要はない．）

(b) その方法を列挙せよ．

（c）　作業が6つのモジュールに分割されたならば何通りの方法があるか？

12.　利息 r で借りた金額 A の金を毎回同じ金額で n 回に分けて返済するときの1回あたりの金額は次の公式で与えられる．

$$P(n) = \frac{Ar(1+r)^n}{(1+r)^n - 1}$$

（a）　加減算，乗算，除算の演算回数 N_A, N_M, N_D を n の関数として表わす公式を導出せよ．$(1+r)^n$ の計算で行なう乗算は $n-1$ 回と考える．

（b）　(a)で求めた公式を使って，n に対する P の表を作成するのに必要な各演算の回数を見積れ．ただし $n = 1, \cdots, 30$ とする．

（c）　次のような P に対する再帰形を導出せよ．

$$P(1) = A(1+r)$$
$$P(n+1) = \frac{P(1)P(n)}{A + P(n)}$$

（d）　再帰形に対して(a)と(b)の問にもう一度答え，非再帰形の場合と結果を比較せよ．

13.　次に示す二項の公式の再帰形を代数的に導け．

$$B(0;\ n,\ p) = (1-p)^n$$
$$B(k+1;\ n,\ p) = \left(\frac{p}{1-p}\right)\left(\frac{n-k}{k+1}\right)B(k;\ n,\ p) \qquad k = 0, 1, \cdots, n$$

$n = 10$ の場合に，B の再帰形と非再帰形との間で，加減算，乗算，除算の回数を比較せよ．

14.　ある計算機システムでは，オンライン・ユーザが6人居る．各ユーザがある時点で存在し得る状態は，次の3つのうちの1つだけである．

状　態	確　率
B（使用中）	1/3
T（考慮中）	1/3
W（待ち）	1/3

（a）　各ユーザの標本空間を示せ．

（b） このシステムの状態には何個の標本点があるか？ システムの標本点の例を少くとも4つ示せ.

（c） “n 人のユーザが使用中である”(ただし，$n = 0, 1, \cdots, 6$)に対する標本空間を考え，標本点の確率を計算せよ.

（d） 使用中のユーザの人数の平均値を計算せよ.

（e） 各ユーザの状態確率が 0.1, 0.3, 0.6 であるとして，(c)と(d)にもう一度答えよ.

15. ある特別な目的のシステムのすべてのユーザは100個のプログラムの集合を利用でき，各ユーザはどの時点でもただ1つのプログラムを(どのプログラムについても同じ確率で)実行していなければならない．10人のユーザが居る場合，2人のユーザが同一のプログラムへのアクセスを要求することがない確率はいくらか？

16. ある計算機システムには記憶装置は2つしかなく，早い方のサイクル・タイムは t_f，遅い方のサイクル・タイムは t_s である．早い記憶装置への参照の確率が p であるとして次の問に答えよ.

（a） システムの平均サイクル・タイムを表わす(簡単な)公式を導出せよ.

（b） 早い記憶装置は，遅い方より10倍早いとする．システムの平均スピードは早い記憶装置の4倍しか遅くならないとするには，各記憶装置へのアクセスの確率はいくらでなければならないか？

17. あるシステムは，CPU，磁心記憶装置，カード・リーダ・パンチャをそれぞれ1台ずつ持っている．これらの構成要素のおのおのは稼動中か故障中かのいずれかであり，ある与えられた期間に稼動中である確率はそれぞれ0.98，0.90，0.85である．その期間中にシステムが故障する確率はいくらか？ (注：構成要素のどれかが故障するとシステムは故障する.)

18. ディスクへのアクセスが駆動装置 A に対して行なわれる確率は0.4であることが測定によってわかった．二項分布を仮定すれば，4回のアクセスのうちで次の場合が起こる確率はいくらか？

（a） 駆動装置 A へのアクセスが2回ある.

（b） A はアクセスされない.

（c） A へのアクセスが少くとも1回ある.

（d） A へのアクセスが少くとも2回ある.

19. ある計算機システムは次のようなタイプの装置を持つことができる．

	信頼度
演算装置，AU	0.95
磁心記憶モジュール，CSM	0.92
I/O チャネル，CH	0.85
ディスク・ファイル，DF	0.80
磁気テープ装置，TU	0.80
カード・リーダ・パンチャ，CRP	0.78

(a)　1台の AU，2台の CSM，2台の CH，2台の TU，1台の CRP から成るシステムの信頼度はいくらか？　システムが稼動するためにはすべての装置が稼動しなければならないものとする．

(b)　(a)の問で TU だけを4台に増やし，システムを動かし続けるにはそのうちのどれか2台が稼動していればよいと仮定すれば，システムの信頼度はいくらか？

20. (a)　問題19で今度は AU が2台あり，システムが稼動するにはそのうちの1台だけが稼動すればよいと仮定して問にもう一度答えよ．その結果と問題19の信頼度との比較を行なえ．

(b)　AU を増やしても信頼度があまり改善されないとすれば，それでもやはり AU を2台にせよと勧める別の理由を論ぜよ．

21. 信頼度はすべて指数関数の形であるとして次の問に答えよ．

(a)　あるシステムの故障までの平均時間(MTF)は14時間である．そのシステムが3時間長生きする確率はいくらか？

(b)　14時間長生きする確率はいくらか？

22. 信頼度関数は指数であると仮定する．ある構成要素の MTF が100時間であれば，次のものの MTF はいくらか？

(a)　2つの構成要素が並列に接続されたもの．

(b)　4つの構成要素が並列に接続されたもの．

(c)　6つの構成要素が並列に接続されたもの．

23. ある計算機は50,000個のトランジスタを含み，MTF は50時間でなければならない．指数分布を仮定すれば，各トランジスタの MTF はいくらでないといけないか？

24. $n-1$ 個の構成要素が待機中であり，おのおのは，故障が起ったときに

待機中の構成要素へ切換えるスイッチを持っているような構成要素待機のシステムを考えよう．各構成要素の信頼度が r，各スイッチの信頼度が w であれば，システムの信頼度は次の式で与えられることを示せ．

$$R = 1 - (1-r)(1-wr)^{n-1}$$

（a） $n=2$ とし，$w = 0.5, 0.8, 0.9, 0.95, 1.0$ のおのおのについて，r に対する R の曲線を描け．

（b） $n = 3, 4, 5$ に対して問(a)に答えよ．

25. 冗長による信頼度改善の代償は，冗長な付加設備のコストである．あるシステムの平均稼動時間は 100 時間，平均修復時間は 4 時間であると仮定しよう．3-22 節のモデルの式(11)，(12)を使い，次の問に答えよ．

（a） システムの信頼度はいくらか？

（b） 2 台のシステムが設置されて 1 台の超システムとして働くならば，その信頼度はいくらか？

26. 3-22 節の式(10)から出発して，n 台の機械のシステムの信頼度を表わす公式を導出せよ．ただし，**修復率**は故障中の機械の台数に無関係であると仮定する．この公式と式(13)とを比較せよ．

参 考 文 献

1. [BA] Barlow, R., and F. Proschan: "Mathematical Theory of Reliability," John Wiley and Sons, Inc., New York, 1965.
2. [BAR] Barron, D.W.: "Recursive Techniques in Programming," American Elsevier Publ. Co., Inc., New York, 1968.
3. [BAZ] Bazovsky, I.: "Reliability Theory and Practice," Prentice-Hall, Inc., Englewood Cliffs, N.J., 1961.
4. [CA] Calabro, S.R.: "Reliability Principles and Practices," McGraw-Hill Book Company, New York, 1962.
5. [DR] Drenick, R.F.: The Failure Law of Complex Equipment, *J. Appl. Math,* vol. 8, p. 680, 1960.
6. [FE] Feller, W.: "An Introduction to Probability Theory and Its Applications," vol. I, (2nd edition), John Wiley and Sons, Inc., New York, 1957.
7. [FL] Flehinger, B.J., and P.A. Lewis: Two-Parameter Lifetime Distributions for Reliability Studies of Renewal Processes, *IBM J. Res. Dev.,* vol. 3, 1959.
8. [FR] Fry, T.C.: "Probability and its Engineering Uses," (2nd edition), D. Van Nostrand Co., New York, 1965.
9. [KE] Kemeny, J.G., H. Mirkil, J.L. Snell, and G.L. Thompson: "Finite Mathematical Structures," Prentice-Hall, Inc., Englewood Cliffs, N.J., 1965.
10. [KL] Kleene, S.C.: "Introduction to Metamathematics," D. Van Nostrand, New York, 1950.

11. [MC] McCarthy, J.: Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part 1, *Comm. ACM*, vol. 3, no. 4, pp. 184-195, April, 1960.
12. [MO] Moore, E.F., and C.E. Shannon: Reliable Circuits Using Less Reliable Elements, *J. Franklin Inst.*, vol. 262, pp. 191, 281, September-October, 1956.
13. [MO1] Moskowitz, F., and J.B. McLean: Some Reliability Aspects of Systems Design, *IRE Trans. PGRQC*, vol. 8, no. 7, 1956.
14. [MO2] Mosteller, F., R.E.K. Rourke, and G.B. Thomas, Jr.: "Probability with Statistical Applications," Addison-Wesley Publishing Company, Reading, Mass., 1970.
15. [NE] Neumann, J. von: "Probabilistic Logics and Synthesis of Reliable Organisms from Unreliable Components," Princeton University Press, Princeton, N.J., 1956.
16. [P1] Pierce, W.H.: "Failure-Tolerant Computer Design," Academic Press, Inc., New York, 1965.
17. [RO1] Roberts, N.H.: "Mathematical Methods in Reliability Engineering," McGraw-Hill Book Company, New York, 1964.
18. [RO2] Rota, G. C.: The Number of Partitions of a Set, *Math Monthly*, May, 1964.
19. [ST] Stoll, R.R.: "Sets, Logic and Axiomatic Theories," W. H. Freeman and Co., San Francisco, Calif., 1961.
20. [WI] Winograd, S., and J.D. Cowan: "Reliable Computation in the Presence of Noise," MIT Press, Cambridge, Mass., 1963.

第 4 章

ジョブ・プロセシング・モデル
——非形式的およびシミュレーション的観点——

現代の工業化社会は，莫大な数のジョブをサービスするシステムで満たされている．ジョブの性質は，工業製品の生産から，銀行業務のトランザクションの処理，計算機センターに委託される業務(自分自身の業務も少しはある)にわたっていろいろに変わる．電話システムで見られるように，いくつかのサービス施設はほとんど完全に自動化されているが，その一方では，銀行の出納係窓口におけるトランザクションのように，主として人手によるものや，さらにいくつかの機械工場のように，人手によるサービスと機械によるサービスとを結合したものも多い．このような違いやその他にもいろいろ違いがあるにもかかわらず，基本的なレベルでは，これらのシステムに共通する点が多い．これらのシステムはすべて，サービスを処理するリソースを少ししか持っていないという特徴がある．“少し”というのは，処理しなければならないジョブの数に比べれば少ないという意味である．おびただしいジョブ・ストリームが，数少ないリソースに集中するので，いつでもというわけではないにしてもほとんどの時間，リソースは使用中であろう．われわれは，ジョブの到着およびサービス要求の特性，またリソースの特性およびリソース管理の戦略と，システムの性能の明確な尺度との間の関係に関心がある．この章の目的は，そのようなシステムの重要でかつ共通した特徴をとらえたかなり理想化されたモデルを使って，それらのシステムに対する非形式的な“実感”を与えることである．

非形式的なアプローチは，常識的な研究法やシミュレーションの方法によって，新しい実際的な問題を解くときに有用である．**5**章では，解析的な方法，すなわち，**ある**システムについての，**ある**性能尺度に関する数学的な公式が導出できるような，形式的なモデルを考察する．しかしながら，たいていの現実のシステム(そして有用な性能の尺度の多く)は，既存の解析的モデルによって扱うことができないということを強調しなければならない．それゆえ，この章

の残りで論じられるいくつかの方法と結果は，5章のものにくらべ，正確さには欠けるけれど，より一般的で，適用範囲もより広いものである．それにもかかわらず，作業負荷，リソース，スケジューリングのパラメータを性能に関係づけるような方程式を生みだす解析的方法は，それが適用可能な場面では，数学的証明を保証するアルゴリズム的な設計手続きを示唆するばかりでなく，システムのふるまいへの深い洞察を与えることができる．

ここで，ジョブ-プロセシング・システムの議論で用いる用語の説明をしておくのがよいであろう．ここでの，“ジョブ”ということばは，他のすべてのジョブと，**論理的に独立な**，ある識別可能な仕事のまとまりを表わすのに使われる．したがって，1つのジョブを行なう仕事は，決して他のジョブの仕事に依存しない．ジョブは，同一のリソースに対し，互いに独立に要求を出すという形でのみ，相互に干渉しあうのである．各ジョブ内には(原則として)，複数の**タスク**があり，このタスクは，互いに論理的な関係を持つ仕事の単位である．しかし，ここでは，特にことわらない限り，各ジョブは1つのタスクから成るものとする．

この章の主な目的は，最も普通に使われる用語を反映した1つの用語体系を使って，いくつかの抽象的モデルを議論することである．後で，タイムシェアリングのようなもっと特殊なシステムを論じるときには，われわれは，**tract** [トランザクション(transaction)の省略形] ということばを，**ジョブ**というもっと包括的なことばと同じ意味で使う．

4-1　システム・モデルの性質

われわれが使う**システム・モデル**ということばの意味は，ジョブという作業負荷が与えられている，リソースが相互に連結された形態の抽象的な表現のことである．システム・モデルの完全な記述は，次に示すようなタイプの情報から成る．

1. 作業負荷の記述
2. システム構造
3. スケジューリング
4. 性能の指標

作業負荷の記述とは，たとえば，ジョブの到着時刻や，ジョブの要求実行時間などのように，ジョブを特徴づける項目を述べたものである．**システム構造**というのは，個々のリソースおよび，ジョブがシステムに入ったり，出て行っ

たり，システム内を動いたりするときの経路を意味している．**スケジューリング**規則は，どのようにジョブを選んでシステム構造内を動かすかを規定するものである；よく見られる例は，FCFS(first-come-first-served, **FIFO** とも呼ばれる)や SXFS(shortest-execution-first-served)である．性能の指標は，システムの“良さ”を測定するための1つまたはそれ以上の方法を定義する．性能の指標は次の2つのクラスに分類される．

1. ジョブ・ストリームの性能
2. リソースの性能

ジョブ・ストリームは次のものを含んでいる：平均の経過時間または応答時間，スループット，いろいろな範囲の実行時間についての応答時間の度数分布，など．この種の尺度は，リソースに対し直接労力を払うシステム・オペレータやシステム管理者よりはむしろ作業負荷を供給するユーザ向きのものである．これに対し，**リソースの性能**は，リソースの使用効率やオーバラップなどの因子を含んでいる．

以後の節では，システム・モデルのおのおのの側面についてある程度詳細に考えていく．

4-2　作業負荷の記述 [BU 2, SR 参照]

すべて性能の研究には，まず作業負荷を記述する方法，そして次に，記述のパラメータの数値を提供してくれるデータが必要である．

記述の方法は，研究の性格に依存する．性能バグを発見するためにシステム開発の後期段階でよく行われたり，受入れ検査用の実行やシステムのセールスマンによる入札の際によく行われたりするように，実際に性能を測定するには，現実のジョブからとった**ベンチマーク** (benchmark) と呼ばれる1つの標本を使うのが普通である．そのジョブはたいてい，考察中のシステムに似たシステムで現在動いているものから選ばれる．ベンチマーク用のジョブを選ぶときの重要な基準のいくつかは，次のものである．

1. 最も頻繁に実行されるジョブ(最も多数のジョブを代表するジョブ)．
2. システムの稼動時間やリソースの大部分の使用の原因となるようなジョブ．
3. ジョブの終了時間に関する要求がシステムの機能を行なう上で最もきびしい条件となるようなジョブ

たいていの場合，これらの基準の1つに最もよくあてはまるジョブの集合は

他の基準にはあてはまらないので，多くの基準によってジョブを選ぶという方法が指示されている．

ベンチマークを選ぶ場合のもう1つの問題は，**スケーリング**(scaling)，すなわち，選ばれたジョブは1カ月あるいは数カ月にわたる動きを反映したものでなければならないけれど，その全実行時間が妥当な長さ(たとえば数時間)であるという制限をつけて，たとえば上記の基準にしたがってジョブの集合を選ぶということである．

ここ数年来，**合成作業負荷**(synthetic workload)という方法に，かなり関心が持たれてきた．この方法では，CPU時間や，いろいろなタイプのI/O時間などのような，システム活動の輪郭が作成される．その輪郭は通常，現実のシステムの活動を分析することによって得られる．しかしながら，テスト用の作業負荷は，輪郭を形成するときに使われた作業負荷と(少なくとも，輪郭の記述子に関する限り)同じシステム活動を引き起こす疑似作業負荷である．合成作業負荷の方法が優れている点は，スケーリングのために作業負荷の実行モジュールを簡単に修正できたり，いろいろな輪郭に対し同一のモジュールが使えたりする点である．

ベンチマークの選択や輪郭の形成などの技術的な問題のほかにも，性能を調べるために作業負荷を形成する際に，別の考慮すべき重要な問題がある．たとえば，ジョブを基にしたものは，慎重に行なえば，現在までのジョブ活動を適切に表現することができるが，しかし，これらは，現在以降のジョブを表わしているだろうか？　もちろん，このタイプの質問には，一般的な技術上の答はない．しかし，管理上の答はある：**システムの分析やシミュレーションを行なう人は，作業負荷の代表として使う現実のジョブ・ストリームについて，(その結果を使うであろう人達から)前もって承諾を得ておかなければならない．**分析者は，作業負荷の重要な構成部分を識別する手助けをすることができるけれども，ここでの主導権は，システム設備の購入や実際の運転に責任のある人達にゆだねられるべきである．また，作業負荷についての承諾は，最終的にはその決定を確証する明確な記録として残されるべきであり，理想的には，どんなモデル化を行なってもその結果が得られる前に，このことを行なっておくべきである．性能の調査を必要としている人達が，ある特定のシステムに対し，意識的なあるいは無意識の偏見を持っていると考えられるならば，上記のことが特に重要となる．というのは，一般に作業負荷を適当に選ぶことによって，調査の結果をどんな方向にも望む方向に曲げることができるからである．

性能システムは通常人間システムに埋め込まれており，この単純ではあるが

意味深長な事実は，ユーザがシステムから受けるサービスに重大な影響を及ぼす．このサービスに直接影響を及ぼす要因の内のいくつかは，次のものである．

1. ジョブから次のバッチを構成するために，その指示された点から個々のジョブ・デックを人手によって取り出すのに要する時間，磁気テープやディスク・パックを装塡したり取りはずしたりする時間，等．
2. 出力を人手で処理するのに要する時間．これには，プリンタ用紙を切離したり，ユーザ用の出力架に出力リストやカードデックを分配したりする時間が含まれる．
3. ジョブ・ストリームを規定する際には，ジョブの到着時刻の情報や，到着順序の情報にさえも，余り注意が払われないことが多い．

これらの要因のうち，最初の2つは理解しやすいが，忘れられがちである．入力ストリーム用のジョブを集めたり，実際のユーザにジョブの出力を分配したりする人手による機構は，システムの機械部分がうまく働いているにもかかわらず，ユーザが感じる不満の大部分の原因になっている．ジョブの順序付けに関する，上記の3番目の項目は，非常にこみ入った問題である．たとえば，n 個のジョブの集まりには，$n!$ 通りもの相異なる順序付けの方法が可能であり，ある特定のジョブ・ストリームのデックは，そのうちのたった1つの方法にすぎない．いったんシステムにストリームが入力されたならば（すなわち，ジョブがバッチに組まれた後では），ジョブの順序の影響が重大なものかどうかは，システムがリソースをオーバラップして使うかどうかとか，スケジューリング処理の特性とか，使われる性能の尺度とかに依存する．スケジューリングが，ジョブの到着順序以外のパラメータに基づいて行なわれたり，スケジューラが，スケジューリングの決定の際に順序にかまわずストリームの走査を行なうことが可能であるならば，元のジョブ・ストリームの順序はそれほど重要ではない．IBM OS/360 のような最近のオペレーティング・システムでは，実際にこのようになっており，OS/360 は，ジョブストリームを走査してそれぞれのジョブの優先順位とクラス番号を見つけ，これに基づいて，ジョブをリソースにスケジュールしている（しかし，タイブレーキング† のためにはやはり FCFS 方式が使われる）．

現実のベンチマーク・ジョブ・ストリームの方法や，合成ジョブの方法の使用は，既存のシステムの性能を検査するのに適している．しかし，リソース

---

† 訳註： タイブレーキング　競技で同点の均衡をやぶること．ここでは，複数のジョブの優先度がある方式で同一とされた時，それらのジョブ間の優先度を決定することを言う．

のさまざまな構成形態に対する性能の依存性の性質を調べたり，計算機性能の理論に寄与したりするためには，作業負荷のモデルを与えてやる数学的モデルやシミュレーション・モデルを構成する必要がある．このようなモデルには，次のものが必要となる．

1. ジョブ i がシステムに到着する時刻
2. ジョブ i のために，おのおののシステム・リソースに要求されたサービス時間
3. ジョブ i がリソースを要求する順序

1番目は1つの数値 A_i，すなわち，ジョブ要求が到着した時刻の消印である．システム全体を1つのリソースと考えるならば，1つの数値 X_i が，ジョブ i のために要求されたリソースの使用時間であり，ジョブ i の順序に関する第3の情報は不要である．X_i の値はシステムの処理速度に依存するということに注意しよう．たとえば，50×50 の行列の逆行列を求めるというジョブをモデル化するための X_i は，モデル化されている特定のシステムでのこのジョブのための時間である．

定　義　　タイプⅠの作業負荷モデルは，各ジョブ i が A_i, X_i の数値の対で特徴づけられ，これらはそれぞれ，到着時刻と要求された処理時間とを表わす．われわれは，A_i のベクトルは，その値が非減少の順で分類されていると仮定する．

タイプⅠの作業負荷モデルは，簡単なものではあるが，十分現実性を持っており有用である．したがって，ジョブ・ストリーム・モデルはベクトル A と X の対から成っており，A と X の対応する成分は，同じジョブに関するものである．タイプⅠのデータを表現するには，(1) 羅列する方法と (2) 確率的な方法の2つの方法がある．**羅列する**というのは，各ジョブに対する到着時刻と実行時間を直接明記するという意味である．**確率的な**というのは，これらの時刻と時間は，そのシステム(または，同じようなシステム)で集められた履歴データから作られた度数分布から，第2-5節で述べた方法を使って生成されるという意味である．Markov 連鎖を使ってタイプⅠの作業負荷を一般化する方法が3-17節に示されている．

タイプⅡの作業負荷表現は，**準備時間**(set up time)も含むようにタイプⅠを一般化したものである．われわれはまた，準備時間に等しい後処理時間も仮

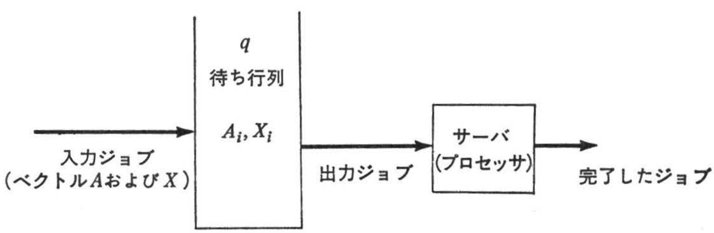
定する.

定　義　　タイプⅡの作業負荷モデルは，各ジョブが3字組 A_i, X_i, S_i で特徴づけられ，これらはそれぞれ，ジョブの到着時刻，実行時間，準備時間を表わしている.

上記の2つの作業負荷記述は，非常に簡単であり，またジョブの**時間**特性だけを扱うものである．計算機システムでは，どのジョブもすべて，高速メモリや補助記憶装置(ディスクなど)内の**空間**(記憶装置)をもまた必要とする．この章の大部分では，簡単化のために，時間のスケジューリングのみに注意を向けよう．また，時間のスケジューリングは，そのかなりの部分が，計算機に無関係な分野で永年にわたり積みあげられてきた理論や経験から成っている．空間のスケジューリングは，主にマルチプログラミング・システムに適用できるものである．われわれは，後にタイムシェアリング・システムや仮想システムを議論するときに，**時間および空間**のスケジューリング・モデルを考察する.

4-3　最も簡単な単一サーバ・モデル

単一サーバ(窓口)モデルは，最も簡単でまた最も広く理解されているものである．それは，すべてのシステム・モデルが持つ特性の多くを示している．(図4-3.1参照)到着時刻と実行時間で特徴づけられるジョブ要求は，最大容量がジョブ q 個である待ち行列の領域に入ると仮定する．待ち行列が一杯である時に到着したジョブはすべて失われてしまい，システムに入らないと仮定する．たいていの場合，特にことわらない限り，q は十分大きくて，どんなに待ち行列が大きくても収容できるものと仮定する．単一の処理装置はいつでも，スケジューリング規則で示される方法に従って，待ち行列から一時に1つずつジョブを選び出して仕事を行なう．今のところは，スケジューラはFCFS方式で



4-3.1　簡単な単一サーバ・モデル

あると仮定する；すなわち，ジョブはその到着順に選ばれてサービスを受け，各ジョブの処理が終了してから，次のジョブの処理が始まる（第4-4節では，他のスケジューリング規則も考察する）.

ジョブは等時間間隔で到着して同じ処理時間を要求するとすれば，モデルの動きや性能の解析はかなりつまらないものとなる．実際の場合にもまたわれわれのモデルでも，ジョブの到着時間間隔や要求処理時間はジョブ毎に異なっているが，そればかりでなく，これらが予測不可能であると仮定することも多い．しかし，ジョブの到着時刻と実行時間を個々に予測することができない場合でも，第4-2節で議論したようにそれらの分布関数は知られているものと仮定する．

図4-3.2は，単一サーバ・システムにおける定義と考察事項の大部分を簡潔な形に要約したものである．読者は，本章を通してこの図を参照されたい．到着時刻と実行時間の分布がどのようなものであっても，それらの**平均**の率に焦点をあてるならば，定常状態という条件の下で，

$$\alpha = \text{平均到着率(ジョブ/秒)}$$
$$\omega = \text{平均実行率(ジョブ/秒)}$$

としよう．

そうすると，次にあげるような互いに重なりのないすべての場合を尽す集合を示すことができる．

場合 1.　$\alpha > \omega$　または　$r = \dfrac{\alpha}{\omega} > 1$

場合 2.　$\alpha = \omega$　または　$r = 1$

場合 3.　$\alpha < \omega$　または　$r < 1$

このように，rは，平均到着率と平均実行率との比である．これらの場合に対する“感じ”を得るには，次のように，流体の流れからの類推で考えればよい．すなわち，ジョブは流体であり，入力パイプからタンクで表わされる待ち行列領域に流入し，別のパイプを通って処理装置へと流れ出して行く．流出と流入のある瞬間の率は予測不能であるが，それらの平均の率はαとωである．したがって場合1.は，平均してみればジョブは処理されうる速度よりも速く到着するということを表わしている．超過したジョブの“流体”はタンクに溜まっていき，したがって待ち行列の長さ（待ち行列に並んでいるジョブの数）と待ち時間は，時間がたつにつれて際限なく増えていく傾向にある．待ち行列のある

ジョブの記述子と性能

n = ストリーム内のジョブの個数
A = ジョブの到着時刻
X = ジョブの実行時間
D = ジョブの終了期限
Q = ジョブの終了時刻
$Q - A$ = ジョブの経過(応答)時間
$Q - D$ = 終了期限からの遅れ
$\bar{t}_e = 1/n \sum(Q - A)$ = 平均経過(応答)時間
$H = n/(\mathrm{Max}(Q) - \mathrm{Min}(A))$ = スループット

リソースの記述子と性能

m = リソースの数(単一サーバの場合には = 1)
T = 総経過時間 = $\mathrm{Max}(Q) - \mathrm{Min}(A)$
u = リソース使用時間
u/T = リソースの使用率
t_j = 待ち行列の長さが j である総時間
$\bar{j} = (\sum j t_j)/T$ = 平均待ち行列長

図 4-3.2 定義と表記法

瞬間の長さはある時点から次の時点の間で短くなることもあるかもしれないにしても，このことは正しい．場合 **3.** では，平均処理率は平均到着率より大きいので，サーバは，少なくとも平均値が定義されている長い時間にわたって見れば，負荷に追随していくことができる．待ち行列の長さと待ち時間は，時々刻々変動するけれども，それらの平均値は，α と ω の関数である有限値でなければならない．これらのすべての結論は，**5** 章のこのモデルの解析的な解において(少なくともある条件の下では)厳密に確証される．

単一サーバの FCFS システムの性能に関するいくつかの簡単な公式が導かれる．まず第 **1** に関心があるのは，あるタイプ I の作業負荷が与えられたとき，到着したジョブの仕事はすべて完了したがまだ次のジョブが到着していないために，サーバに遊びの期間が生じることがあるかどうかということである．n 個のジョブのストリームに対し，遊びの時間が生じない条件は；

$$A_i \leq A_1 + \sum_{j=1}^{i-1} X_j \qquad i = 2, 3, \cdots, n \tag{1}$$

これらの不等式は，以前に到着したジョブがすべて実行されてしまってから到着するようなジョブはないということを保証している．非常に簡単で特別な例

は，すべてのジョブが，たとえば時刻0すなわち $A_i=0$ に同時に到着する場合である．この場合には，ジョブ i の経過時間は，ジョブ i およびそれ以前のジョブの実行時間の総和となる．

$$Q_i - A_i = Q_i = \sum_{j=1}^{i} X_j \tag{2}$$

したがって平均経過時間は，X_j の部分和の総和を n で割ったものとなる．

$$\bar{t}_e = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{i} X_j \tag{3}$$

式(3)は，ある与えられた X_j の任意の順列に対して平均経過時間を計算するときに有用である．n 個のジョブの集まりが与えられると，$n!$ 通りの並べ方が可能であり，そのおのおのは，独自の $\bar{t}_e$ の値を持った相異なるジョブ・ストリームである．これらの平均値の，可能な $n!$ 個の順列のすべてにわたる平均値は，次の式で与えられることがわかる．

A	0	0	0	0	0
X	12	4	3	5	2
Q	12	16	19	24	26
$t_e = Q - A$	12	16	19	24	26

平均経過時間：$\bar{t}_e = \dfrac{\Sigma Q - A}{n} = \dfrac{97}{5} = 19.4$

(a) 同時到着の場合

A	1	2	18	20	22
X	12	4	3	5	2
Q	13	17	21	26	28
$t_e = Q - A$	12	15	3	6	6

平均経過時間：$\bar{t}_e = \dfrac{\Sigma Q - A}{n} = \dfrac{42}{5} = 8.4$

(b) 到着時刻にずれがある（遊び時間がある）場合

図 4-3.3　ジョブ・ストリームの表現(FCFS 単一サーバ)

$$\bar{t}_e = \left(\frac{n+1}{2n}\right)\sum_{i=1}^{n} X_i \qquad (4)$$

図 4-3.3 は，簡単なジョブ・ストリームの例について示したものである．図の(a)は，5個のジョブが同時に到着した場合であり，したがって式(1)と(3)が成り立つ．

4-4　スケジューリング

性能の唯一の“正しい”目標関数というものがあるわけではない：システムの設計目的であるサービスの種類によって，多くの判断基準が存在しうる（しかし，考えうるもののいくつかは，実現不可能である）．“最適”について正確に述べれば，そのおのおのが別々のスケジューラになるであろう．

定　義　スケジューラとは，どのジョブが，リソースによる次のサービスを受けるべきであるかを，ただ1つだけ明確に規定するアルゴリズムである．

ある明記された，“良い性能”の目標関数が最適となるように，ジョブの選択が行なわれる．

次にあげる慣用記号を使えば，いくつかの特定なスケジューラの議論が簡単になる．

V = スケジューリング・ベクトル；V_i は，ジョブ i に対するスケジューリング変数の値を示す．
n = システム内にあるジョブの個数（$= \rho V = V$ 中の値の個数）
j^* = スケジューラによって選ばれたジョブ

最適化は，いつでも次のような最小化として表現される（最大化としてもよかったのではあるが）：

$$j^* = \underset{i}{\text{Min}}(V_i) \qquad i = 1, 2, \cdots, n \qquad (1)$$

2つの特別な例は：

FCFS スケジューラ　このスケジューラは到着順にジョブを選ぶので，スケジューリング・ベクトル V は，到着時刻ベクトル A である．

SXFS スケジューラ　このスケジューラは，実行時間が最小のジョブを1番に選ぶので，スケジューリング・ベクトル V は，実行時間ベクトル X である．

V_i は，上記の例のように1つの変数であっても，後で例が示されるように1つまたはそれ以上の変数の決められた関数であってもよい．いずれにしても，スケジューラとは，式(1)を使って V の値を求める手続きであるということで定義される．しかしこれではまだ，スケジューラの完全な定義とは言えない．というのは，式(1)の結果は**同点**(tie)となり得るので，サービスを受ける次のジョブがただ1つだけ選択されるということは必ずしも保証されていないからである．2個以上のジョブが同一の V_j の最小値を持つ場合に，これが起る．たとえば，FCFS スケジューラの場合に，到着時刻であるスケジューリング・ベクトルが次のようになっているとしよう．

$$V = A = 2, 2, 2, 4, 4, 7, 8$$

そうすると，式(1)から V の最小値は2であり，時刻2に到着した3個のジョブが同点になり，j^* はベクトルになってしまう．

$$j^* = 1, 2, 3$$

定義により，完全なスケジューリング・アルゴリズムはいつでも**ただ1つ**のジョブを選び出さねばならず，同点の可能性があれば，本来の選択規則だけでなく，あるタイブレーキングの規則も含んでいなければならない．タイブレーキングの1つの簡単な方法は，同点ベクトル j^* で最初に現われるジョブを選ぶことである．もう1つの方法は，ベクトル j^* で示されたジョブに，FCFS 規則を適用することであるが，これによってもまた，同点の生じることがあるということに注意しよう．サービスを受けるものとして最終的に選ばれた唯一のジョブは，“最も高い優先度”を持っていると言われる．

重要で一般的な，スケジューラを分類する1つの方法は，それらが**横取り的**(preemptive)であるかどうかによる分類である．

定　義　　**横取り方式のスケジューラとは，必ずしもジョブの完了時刻でないときでもジョブを選択し，サーバへの割当てを行なうことができるものである．**

暗黙のうちに，横取り的でないスケジューラは，ジョブが終了した時点でのみ，ジョブ選択と割当てを行なうものとされている．このように，横取り方式のスケジューラでは，ジョブはすべて**割込み可能**であり，実際にあるジョブに割込んで，リソースが優先順位の最も高くなった別のジョブのサービスを行なうこ

とを許すものと仮定する．このことは，横取り方式のスケジューラでは，ジョブのスケジューリング変数の順位はジョブの実行が開始された後でさえも変更可能である，ということを暗に意味している．また，リソース(サーバ)は，ジョブが途中で割込まれても，ある不定時間後に完了され得るようにするのに必要なジョブに関する情報を入手できなければならない．割込み使用は，スケジューリングの柔軟性の改善をまねき，この柔軟性は本来，性能改善に資する．しかしながら，サーバがジョブを切り換えるたびに，準備のための“オーバヘッド”時間という代償をしばしば払うことになる．機械工場では，準備とは，初めに道具や仕事を整頓することである．計算機システムにおける準備とは，典型的には，レジスタや高速メモリの一部に対するロードやストアを行なうことである．横取り方式でないスケジューラに対しては，この準備はジョブ1個につき1回行なわれるだけだが，横取り方式のスケジューラに対しては，準備のオーバヘッドは**ジョブの割込み毎**に2回ずつ(割込み時に1回，再開時に1回)必要である．準備時間のオーバヘッドは，V_i をこの時間の関数とすることによって，スケジューラに含められてもよい．

横取り方式のスケジューラがジョブ切り換えの必要を認めたときには，ジョブ切り換えに伴なう準備時間のオーバヘッドを考慮して，次のスケジューリングの決定を考える前に，切り換えられた新しいジョブに，処理装置をある最小時間使わせてやることが得策な方法である．この短い時間単位は，**量子**(quantum)と呼ばれ，計算機システムでは，典型的には100ミリセカンドのオーダーである．このように，スケジューリングが行なわれるのは，“量子の終り”かジョブの終りの時点だけである．

たとえば，**LCFS**(least-completed-first-service)**スケジューラ**は，ジョブの実行時間のうち実行済みの部分が最も少ないジョブが一番優先度が高いものであると説明される．したがって，新しいジョブ(実行済みの時間が0である)には自動的に最高の優先順位が与えられ，新しく到着したジョブに対する仕事を**いくらか**行なうために，スケジューラは現在進行中のいかなる仕事にでも割込みをかける．スケジューラはまた，あるジョブの仕事が進んでその実行済み部分が，別のジョブの仕事のうち実行済み部分が最小のものより大きくなると，ただちに実行中のジョブに割込みをかける(このスケジューラについてのもっと詳しい議論は下記を見よ)．

さてこれから，簡単ではあるが有用なスケジューリング・モデルにおける，主な考慮事項について述べよう．特定のスケジューラをもっと良く議論するために，**図4-4.1(a)**はスケジューリングの関数にしばしば現われる6つの重要

A_i = ジョブ i の到着時刻
X_i = ジョブ i の実行(要求されたサービス)時間
D_i = ジョブ i についての希望の“終了期限”
R_i = ジョブ i に対する静的な優先順位(最低値 = サービスの優先権が最高)
P_i = ジョブ i についての実行ずみ時間($P_i \leq X_i$)
L_i = ジョブ i の最後のサービス期間(量子)が開始された時刻

（a）スケジューリング変数

スケジューラ	$\mathbf{Min}_i(V_i)$ に対するスケジューリング変数	注
FCFS	A_i	First-come-first-service
SXFS	X_i	Shortest-job-first
LCFS	P_i	Least-completed-first
EDFS	D_i	Earliest-due-time-first
HSFS	R_i	Highest-static-priority-first
RR	L_i	Round robin(RR)

（b）基本的なスケジューラ

図 4-4.1　スケジューリング変数と基本的なスケジューラ

な変数を列挙したものである．ジョブがシステムに入るとただちに，これらの変数のうちの **3** つ(到着時刻，終了期限時刻，静的な優先順位)を知ることができる．要求される実行時間 X_i は，その時点でわかっていることもあればそうでないこともある．計算機システムでは，X_i が正確に知られていることはほとんどない．というのは，X_i はしばしば，ジョブ自身によって計算されるいくつかのデータの値に依存し，したがってジョブの実行前にこれを知ることはできないからである．ユーザやジョブの過去の履歴によって見積もられた実行時間が使われることもある．しかし，ジョブの実行時間の明確な評価値がない場合でさえも，スケジューラは短いジョブに都合のよいように定義できる（下記を見よ).

2つの変数(P_i と L_i)は，システムの進行につれてその値が変わる．P_i は X_i のうちすでに完了した部分であり，L_i は，ジョブ i が**最後の**量子を開始した時刻である．これら2つの変数は，横取り方式のスケジューラでのみ有効である．

図 4-4.1(b) は6つの基本的なスケジューラを列挙したものである（それぞれのスケジューラは，スケジューリング変数として(a)の **6** つの変数のうちの **1** つを使うという意味で基本的である)．最初の **3** つのスケジューラ（FCFS,

SXFS, LCFS)についてはすでに述べた．今から，他の3つについて簡単に説明を加えよう．

EDFS(earliest-due-time-first)**スケジューラ**は，各ジョブについて希望する終了期限の宣言を必要とし，終了期限が最も早いジョブを次にサービスするジョブに選ぶものである．**HSFS**(highest-static-priority)**スケジューラ**では，各ジョブに対して，入力時に先立って静的な(固定の)優先順位数が宣言されており，これらの数値がスケジューリング・ベクトルになる．これら2つのスケジューラは両方とも，ユーザが直接スケジューリングを制御するという印象を与えるので，単純化しすぎだといわれがちであるけれども，これらはいくつかの落し穴を含むことがある．もしユーザが完全に自由に自分のジョブにスケジューリングの数値を与えてもよいならば，スケジューラは事実上，タイブレーキングの規則に“退化”してしまうことがしばしばある．したがってユーザは，スケジューリング変数の値を自分にとって最も好都合な値に設定することによってのみ，よいサービスを受けることができるということを発見する．また一方で，もしユーザに規律が課せられて優先順位や終了期限時刻を自由に規定することが制限されるならば，この規律の規則はまさにスケジューラの一部に他ならず，それは自動的なスケジューリング手順を外側から見た姿となる．1つの素朴なやり方は，ユーザは自分が本当に必要とするサービスのクラスのみを保証するパラメータを規定するという，ユーザの“好意と協力”にたよることである．しかしそのようなやり方では簡単に失敗してしまう，というのは，人間には，他の人より優位にたとうとする(常にそうする人が何人かいる所では特に)性癖があるからというだけでなく，その方針は，よいシステム設計の基本原則，すなわち，ユーザの間の衝突(この場合にはリソースの使用をめぐる衝突)を解決する責任を持つのは**ユーザではなくてシステムである**という原則を無視しているからである．個々のユーザは当然のことながら彼ら自身のジョブを案じており，システム全体を案じるための知識も動機も持ちあわせていないのであるから，システム設計の原則はこうでなければならない．次のような性能の保護原則を述べることにより，この点を強調しよう．

システムは，どれか1つの(あるいは2, 3の)ジョブにより，たとえどんなスケジューリング情報が与えられたとしても，ジョブ・ストリーム全体にわたって，妥当なよいサービスを保証するように，設計されなければならない．

スケジューラの簡単ではあるが有用な1つのテストは，テスト用に決められた1つのジョブ・ストリームの中からあるジョブをとり出し，システムの性能をわざと低下させようと意図して，このジョブに対して，合法的に(しかし悪意を持って)決められたスケジューリング変数を与えてやることである．うまく設計されたシステムであれば，その同じジョブに好都合なパラメータを選んだ場合と比べても，ジョブ・ストリーム全体にわたっての性能の低下は少ししか見られないであろう．（これは，どの1つのジョブのリソース要求もストリームを支配しないことを仮定している.）

今まで，ユーザによって宣言されたスケジューリング・パラメータに依存するスケジューラについて議論してきたが，もし，そのような情報を利用するスケジューラが性能を保護する予防策をとっているならば，それらのスケジューラはだめとはいえない．たとえば，システムはユーザにジョブの実行時間を見積ることを要求し，これらの時間を使って，システムは短いジョブを最初に実行するようにスケジュールしようとすると仮定しよう．明らかに，ユーザのジョブの本当の長さはどうであろうとも，すべてのユーザは，自分のジョブは短いと宣言したいという強い誘惑にかられる．このことを考慮に入れたやり方も可能であって，それは，もしジョブが宣言された見積り時間をある許容時間以上超過すると，そのジョブは打ち切られる(しかし，それでも課金される)ものである．このような方式は(実際によく使われており)無責任なユーザの宣言に対して，必要な抑制刺激を提供してくれる．このやり方ではしばしば，悪気なく時間を過小評価してしまうユーザに対して不公平になることに注意しよう．肯定的な意味で言うと，スケジューラにとって非常に役立ち，しかもユーザの協力なしに自動的に決めることはまったくむずかしいような，自分のジョブに関する何らかの事柄をユーザは知っているわけであるから，ユーザの宣言はスケジューリングの役に立ち得るであろう．心掛けるべきことは，スケジューリング用に使う前にこの情報の正確さを確保して，またこの情報の悪用を防ぐことである．

しばしば表明されるスケジューリングの目的とは，どのジョブでも，スケジューラに粗末に扱われて“締め出し”を食う，すなわち，不公平に長い時間待たされるようなことがあってはならない，ということである．もちろん，ここでの鍵となる言葉は，“不公平に”というあいまいな言葉である．これから，いくつかの例を使って，この言葉はどんな意味であるかを理解することに努めよう．まず，先に到着したジョブは後から到着したものより遅らされることはないということを保証するFCFSスケジューラを考える．明らかにFCFS

は，ジョブの到着時刻に関してはジョブに対し“公平”であるが，新しく到着した(たとえば)短いジョブに対しては，それより前に到着した長いジョブに比べると“不公平”であると考えられる．もう1つの例として，実行時間の短いジョブがあるとき，実行時間の長いジョブの後の方の部分に対しては，非常に少しのサービスしか与えないLCFSスケジューラを考えよう．その結果は，長いジョブが非常に長時間遅らされる場合が起り得る．

もちろん，われわれは，システム設計者に次のように言うことによって，この不平に答えることができる：「あなたの目標関数は何ですか．それが，あるスケジューラでは満足されないならば，あなたにとってのよいサービスの目標を反映する，別のスケジューラを見つけなさい.」しかし，われわれはどのようにすればこの目標に到達するだろうか．これは奥深い問題であり，後でいくつかのシステムに対して論じられる．われわれは今のところ，スケジューリングの目標は，技術的な因子のみならず心理的な因子も含むことが普通であるので，導出したり正当化したりするには複雑である，とだけ言っておこう．直観的で実用主義的なやり方では，単純なスケジューラの欠点のいくつかを補うためには，しばしば，それらのスケジューラを2つまたはそれ以上組み合せることが望ましい．たとえば，LCFSスケジューラの場合，長いジョブが過度に遅らされるという欠点は，次のようにして取り除くことができる．ジョブiが最後にサービスを受けた時刻を記録する変数L_iを監視する．そして，もし

$$C = \text{現在時刻}$$
$$C - L_i = \text{最後のサービスからの経過時間}$$

とするならば，いかなるジョブiでもその$C - L_i$の値がある決められた値hを超えれば最高の優先度が与えられるように，LCFSスケジューラを修正することができる．(そして，このジョブが1-時間量子の間サービスされる.)そうすると，スケジューリング変数は次のような関数となるであろう．

$$V = P - ((C - L) > h) \times 10{,}000 \qquad (2)$$

$C - L$がhを超えると，不等式が満足されて$(=1)$，10,000という大きな数がPから減じられ，したがって小さなVの値を保証する．

密接なマン-マシン・インタラクションを必要とする多くのシステムの中心になっているサービス原理の1つは，たとえジョブの実行時間がスケジューラに知られていなくとも，実行時間の短いジョブは，長いジョブよりも優先した取扱いを受けるべきであるということである．このような情況下でも良いサー

ビスを供給するための一般原則は，スケジューラが，実行時間の等級付け(順位)の見積りを事実上行なうことである．そのような見積り手順の一例は，実行時間(X_i)の**等級付け**は，実行時間のうち完了済みの部分(P_i)**の等級付け**と同じであるとすることである．したがって，すでに長時間実行されたジョブは，それまでに短い時間しか実行されなかったジョブに比べると，残っている実行時間が長いと見積られる．明らかに，このやり方は，前に議論したLCFSスケジューラにほかならない．

ラウンド・ロビン(RR)・スケジューラもまた，実行時間に関する知識なしに，短いジョブにサービスをよくしたいという要求を満たすことができる．それは，締め出しという問題点がない点で，LCFSより優れている．その基本的な考えは，サービスの終っていないジョブのおのおのに1-時間量子を次々と与え，最後のジョブをサービスした後で，最初のジョブに戻って次の一回り(ラウンド)を始めるということである．式(1)の表現を使えば，スケジューリング変数は，最後の量子の時刻 L_i である(同点となる可能性はない)．

$$j = \operatorname*{Min}_{i}(L_i) \tag{3}$$

しかし，ジョブが到着したときに，どのようにしてそれに L_i の値を割り当てるかを述べなければ，スケジューラのこの記述は，完全ではない．1つのやり方は，L の最大値よりほんの少し大きい値を割り当てることである；この結果，新しく到着したジョブは，次の量子を待っている行列の最後に置かれる．この逆のやり方が，後でわれわれが採用するやり方であって，次のようなものである．すなわち，新しく到着したジョブの L_i には低い値(0)を割り当て，したがってそれが直ちに自分の**最初の**量子を受けることを保証する．この方法をとるRRは，2つの待ち行列――優先度の高い，新しく到着したジョブ用の待ち行列と，少なくとも1回は量子サービスを受けたジョブをすべて含んだ待ち行列――を持つシステムと考えられる．すべてのジョブに1量子ずつ割り当てて，引き続き次の繰返しへ戻るというラウンド・ロビン処理の結果，たとえシステムがジョブの長さあるいはその順位さえ知らないとしても，短いジョブが最初に終了するであろう．式(2)はまた，“部分的な”RRスケジューラであるとも理解することができる．

同じ時刻に到着したジョブのストリームに対しては，LCFSスケジューラとRRスケジューラとでは，同じ結果になる．しかし，これらのスケジューラは，新しく到着したジョブの取り扱い方が異なる．両方とも，新しく到着した

ジョブの**最初の**量子(タイム・スライス)には，優先的な待遇を与える．LCFSでは，新しく到着したジョブは次々とタイム・スライスを与えられ，その合計が，先に到着していてすでに受けたサービス期間が最少のジョブよりも少しでも多くなるまでは，引続きタイム・スライスを与えられる．RR スケジューラの方は，新しく到着したジョブに最初のタイム・スライスを与えると，次は別のジョブに切り換える．

4-5　ジョブの性能尺度

ジョブまたは**ジョブ-ストリームの性能**は，システムに到着してからジョブがこうむった遅れ(待ち)についてをわれわれに教えてくれる．**リソースの性能**(第4-6節参節)は，ジョブ・ストリームを処理しているリソースの使用率を測定したものである．

図 4-5.1 は，各ジョブを説明するいくつかの単一変数から始めてジョブ・ストリーム全体を特徴づける尺度へと，性能の尺度のいくつかを要約したものである．Q_i はジョブ i が終了したときの“壁掛時計”の時刻であるので，その

各ジョブについてのパラメータ

$$Q_i = \text{ジョブ完了時刻}$$

$$Q_i - A_i = \text{ジョブ経過時間}(=\text{応答時間})$$

$$(Q_i - A_i) - X_i = \text{ジョブの待ち時間}$$

$$(Q_i - A_i)/X_i = \text{引き伸ばし率}$$

単一数値のジョブ-ストリーム・パラメータ

$$\bar{t}_e = \frac{1}{n}\sum_{i=1}^{n}(Q_i - A_i) = \text{平均経過時間} = \text{平均応答時間}$$

$$\bar{t}_f = \frac{1}{n}\sum_{i=1}^{n}(Q_i - D_i) = \text{終了期限からの平均遅れ時間}$$

$$H = \frac{n}{\mathrm{Max}(Q) - \mathrm{Min}(A)}$$

$$F = \frac{n}{\sum_{i=1}^{n}(Q_i - A_i)/X_i} = \text{短いジョブのフィギャー-オブ-メリット}$$

分布関数

$X_i/Q_i - A_i$ の相対累積度数分布

図 4-5.1　いくつかのジョブ-ストリーム・パラメータ

ジョブの到着から終了までの**経過時間**は $Q_i - A_i$ である．この時間のうち，X_i が実行時間であり，したがって $(Q_i - A_i) - X_i$ が，ジョブがサービスを待つのに費やした時間となる．**平均経過時間**は，しばしば**平均待ち時間**または**平均応答時間**とも呼ばれるもので，ジョブの経過時間の平均値である．

$$\bar{t}_e = \left(\frac{1}{n}\right)\sum_{i=1}^{n}(Q_i - A_i) \tag{1}$$

ジョブストリームの**スループット**は，ストリーム内のジョブの個数を，ストリームをサービスするのにかかった総経過時間で割ったものと定義される．したがって，単一サーバ・モデルでは，遊び時間も準備時間もなく，スループットは次のようになる．

$$H = \frac{n}{\sum_{i=1}^{n} X_i} \tag{2}$$

もっと一般的に，遊び時間と準備時間とが両方ともは0でない時は；

$$H = \frac{n}{\mathrm{Max}(Q) - \mathrm{Min}(A)} \tag{3}$$

スループットは，平均経過時間の逆数（ジョブ/秒）と同じ単位で測定されるけれども，平均経過時間の逆数には等しく**ない**ということは注意するに値する．たとえば，長さがともに5秒の同時に到着する2つのジョブから成るストリームを考えよう．そうするとスループットは0.2ジョブ/秒となるが，一方平均経過時間は7.5秒でその逆数は0.133である．

式(2)をひと目見ただけで，**スループットは**ジョブの順序づけ，すなわちスケジューラには**無関係である**ことがわかる†．これと際立って対照的に，ジョブの長さが同じでない場合には，平均経過時間はジョブの順序にきわめて敏感である．特に，ある与えられたジョブ・ストリームの平均経過時間は，サービスの順序づけがSXFS(shortest-job-first)のときに最も小さくなる．これは，次のことに注目すればわかる．すなわち，他のどの順序づけでも，長いジョブが早い時間に実行され，この長いジョブの実行のせいで，後の方にスケジュールされたジョブはすべて遅らされてしまう．

短いジョブのフィギヤ-オブ-メリットは次のように定義される．

---

† ジョブの切り換えにはまったく時間がかからないと仮定する．

$$F = \frac{n}{\sum_{i=1}^{n} (Q_i - A_i)/X_i} \tag{3}$$

F は，システムの実行時間の短いジョブによいサービスを与える能力に敏感な尺度である．これを確かめるために，まず原子項 $(Q_i - A_i)/X_i$ を考えよう．これは，**引き伸し率**(stretch factor)と呼ばれる．なぜなら，ジョブ i の可能な最小経過時間 (X_i) はこの率だけ“引き伸されて”，実際の経過時間 $(Q_i - A_i)$ になるからである．F の分母には引き伸し率の総和が現われているので，引き伸し率が小さいと，F の値は大きく(良く)なる．また，短いジョブの場合には，引き伸し率の分子は処理中のすべてのジョブに依存するものであるが，一方分母の方は単に 1 つのジョブの実行時間である．システムが，短いジョブに優先権を与えるのが上手であればあるほど，F の値が良くなる．最後に，F は次のように正規化されていることを示すのは簡単である．

$$F \leq 1 \tag{4}$$

なぜなら，F の分母に現われている各引き伸し率の最小値は 1 だからである．

上で述べた式(1)から(3)までの尺度は単一の数値である．性能，少なくとも実行時間に関する性能についての詳細は引き伸し率の**逆数**の相対度数分布あるいは相対累積度数分布を使って示される．引き伸し率をこの形で表わしたものは，その値が常に 0 から 1 までの範囲にあり，またすべてのシステムに対して 100–バケットの分布表を作ることができるので，便利な形である．そのような分布関数を使えば，次のような重要な質問に直ちに答えることができる：（たとえば）5 以下の引き伸し率で実行が完了したジョブの個数は，全体の何パーセントであるか？

上記の 3 つのジョブ・ストリームの性能尺度は，今までに使われてきたいくつかのものの一例にすぎない．それぞれの尺度は，性能を異なった見地から測定するものであり，また，われわれが 1 つの例で見たように，ジョブ–ストリームを変えても，他の尺度に影響なしに 1 つまたは 2 つの尺度を変更するだけでよい．これらの効果は，たとえオーバヘッド(準備時間)がないときでも，現われる．たとえば，RR スケジューラの方がしばしば FCFS よりも良い平均応答時間が得られる理由は，スケジューラに帰因している：これは，両者ともにオーバヘッドが 0 のときに正しい．

ジョブ切り換え時間のオーバヘッドが 0 ならば，RR と FCFS はともに，スループットが同一になる．オーバヘッドが 0 でないと，FCFS スケジューラ

の方が RR よりも良いスループットを示すであろう．なぜなら，前者は横取り方式でなく，**1個のジョブごとに**1回の準備/後処理を伴なうだけであるが，一方 RR は，**1個の量子ごとに**1回の準備/後処理が必要だからである．また一方で，もしストリームの前の方で長いジョブが現われるならば，FCFS では当然のことながら(ジョブの順序づけがまずいために)平均応答時間が悪くなるが，一方 RR の場合には，平均応答時間はジョブの到着には比較的鈍感である．簡単に要約して言うと，すべてを包括するような性能尺度というものはない；要求されたサービスのタイプによって，いくつかの尺度が有用なものとなる．

図 4-5.2 は，いくつかのスケジューラに対するいくつかの性能尺度の解析的な公式を示したものである．

4-6　リソースの性能尺度：ハードウェア・モニタ
［DR, MU, PA, RO, SC 参照］

ジョブの性能に対してリソースの性能は，ジョブ・ストリームよりむしろリ

	ジョブ j の完了時刻 Q_i	平均応答時間 t_e	スループット H
FCFS	$\sum_{j=1}^{i} X_j$	$\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{i} X_j$	$\frac{n}{\sum_{j=1}^{n} X_j}$
SXFS	$\sum_{j=1}^{i} X_j^1$	$\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{i} X_j^1$	$\frac{n}{\sum_{j=1}^{n} X_j^1}$
ラウンドー ロビン*	$\sum_{j=1}^{i} X_j^1 + (n-i)(X_i^1-1)$	$\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{i} X_j^1 + \frac{1}{n}\sum_{i=1}^{n}(n-i)(X_i^1-1)$	$\frac{n}{\sum_{j=1}^{n} X_j}$

前提：
1. すべてのジョブは同時刻に到着する．
2. X^1 = 値の増加順に並んだ X.
3. *ラウンドーロビン方式は，X^1 の順序づけおよび量子 = 1 を前提としている．
4. 準備時間，後処理時間は 0

図 4-5.2　(特別な条件下での) 3つのスケジューラの解析的比較

ソースの使用についてを中心に扱う．したがって，リソースの性能は，性能の改善をめざす人達のみならず，システムのオペレータ，管理者，購入者の主要な関心事である．システムの個々のユーザやプログラマはそれにはあまり関心がない．おもに関心があるのは，次の事項である．

1. 各リソースの**使用効率**は，使用効率ベクトル U にまとめると便利である．リソース i が使用された時間の合計を u_i，ストリームの処理時間の合計，すなわち，その最初のジョブが到着してから最後のジョブが完了するまでの時間を T とすると；

$$U_i=\frac{u_i}{T}=\frac{u_i}{n}H$$

この定義によれば，使用効率が低いということの原因は，リソースがあまり使われなかったか，あるいは到着時間間隔が長かったかのいずれか（または両方）である．

2. 各リソース上での**待ち行列の最大長**は，システム・パラメータである．なんとならば，システムのコストの一部は，行列に並んでいるジョブを収容する設備を用意するためのものだからである．
3. **オーバラップ**部分は，各リソースと他のリソースが同時に使用された時間 T_B の割合を与えるものである．

使用効率やオーバラップの情報は，現実のシステムではハードウェア，あるいはソフトウェアの手段によって測定される．**ハードウェア・モニタ**は，典型的には**図 4-6.1** で示されるような形をとる．ハードウェア・モニタは，計算機システムの回路に電気的な影響を与えないように慎重に設計された，1組のプローブを用いる．プローブが接続されるシステムの端子は，モニタされているリ

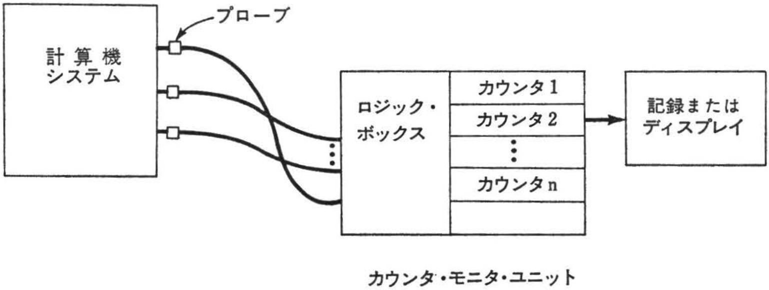


図 4-6.1 典型的なハードウェア・カウンタ・モニタ

ソースの状態(たとえば，“CPU が待ち状態にある”とか“チャネル A が使用中である”など)を示す端子である．モニタされた信号は，典型的には on/off ゲートであり，ロジックボックスを通ってカウンターに送られる．ロジックボックスにはふつう“プラグボード”(plug board) が取りつけられていて，モニタされた信号を論理関数(AND, OR, NOT など)に従って組み合せるようになっており，その結果の信号がカウンターに送られる．たとえば，1つのプローブで“チャネル A が使用中”を，別のプローブで“チャネル B が使用中”を感知すれば，A または B が使用中の時間を計るために，これらの信号は OR される．モニタされているシステム状態は，タイミングパルスのカウンタへの入力を制御するゲートとして使われ，カウンタは，モニタされている事象についてのみパルス(したがって，時間)を数える．カウンタの出力は目に見えるようにディスプレイされたり，人手によって読みとられたり，あるいはまたハードコピーをとるために記録装置へ送られたりする．

ハードウェアモニタは市場で売られたり，またいくつかのシステムの場合には，システムの製造会社から，相談サービス込みで，限られた期間貸し付けられたりする．そのようなモニタは，取り付けが簡単で，また，プログラミングやオペレーティングシステムに何の変更も要らない．たとえメーカが異なるいろいろな異なる機種に対しても，同じモニタ機械が使われる．(ソフトウェアモニタに比べて)ハードウェアモニタが不利な点は，それは，資源の活動状況をジョブのある特定の細かな活動状況に関連づけるには不便だという点である．これは，カウンタが積分器だからである；単一のカウンタの値から，カウントされた細かな事象を回復させることはできない．このように，ハードウェアモニタは，大ざっぱな数値として，リソースの使用効率やオーバラップについてわれわれに教えてくれる．それゆえ，ハードウェアモニタは，たとえば1つの I/O チャネルばかりよく使われてもう1つのチャネルは大部分の時間遊んでいるというような性能バグに焦点をあてる手助けをするのに貴重なものとなり得る．しかしながら，その原因であるジョブの特徴やシステムの特性を正確に指摘するには，もっと詳しい調査が必要になる．

カウンタあるいはソフトウェアを使って使用効率とオーバラップ時間を得ることができると仮定し，われわれはこれから，これらの値を関係づける1つの簡単で優雅な方法を考えることにする．一例として，各リソースはどんな時点でも，2つの状態，すなわち，$I=$ 遊び，あるいは $B=$ 使用中のうちの1つ(しかも1つだけ)の状態をとり得るものと仮定する．特別な例として(**図 4-6.2** 参照)，C と呼ばれる1つの CPU と，P および Q と呼ばれる2つの I/O チ

ャネルを持ったシステムを考える．そうすると，われわれは可能な8つの状態をすべて列挙した表を書くことができる．この場合には，各状態は分離しており，また全体で可能な時間をすべて尽くしているので，図に示されているいろいろな組み合せにおける状態時間の和をとれば，どんな望みの使用効率やオーバラップ時間でも表現することができる［図 4-6.2(b)参照］．

今まで述べてきたシステムやジョブの性能尺度はすべてシステムのみならず**作業負荷**にも依存していることを了解することが最も重要である．それゆえ，

	状態 C	状態 P	状態 Q	状態時間	意味
0	I	I	I	t_0	システムが遊びの状態
1	I	I	B	t_1	チャネル Q だけが使用中(オーバラップしていない Q 時間)
2	I	B	I	t_2	チャネル P だけが使用中(オーバラップしていない P 時間)
3	I	B	B	t_3	チャネル P と Q だけがオーバラップ
4	B	I	I	t_4	CPU だけが使用中(オーバラップしていない CPU 時間)
5	B	I	B	t_5	CPU とチャネル Q だけがオーバラップ
6	B	B	I	t_6	CPU とチャネル P だけがオーバラップ
7	B	B	B	t_7	CPU，チャネル P，チャネル Q がオーバラップ

状態：$I=$ 遊び，$B=$ 使用中

（a） 状態の表とタイミング

CPU が使用中の時間

$$T_C = t_4 + t_5 + t_6 + t_7$$

チャネル P が使用中の時間

$$T_P = t_2 + t_3 + t_6 + t_7$$

チャネル Q が使用中の時間

$$T_Q = t_1 + t_3 + t_5 + t_7$$

CPU がチャネルのどれかとオーバラップする時間

$$T_{C,\,\mathrm{I/O}} = t_5 + t_6 + t_7$$

観測時間の総計

$$T_T = t_0 + t_1 + t_2 + t_3 + t_4 + t_5 + t_6 + t_7$$

（b） 式の例

図 4-6.2 オーバラップと使用率の計算の説明

使用された作業負荷についての明確な記述なしに，スループットや F の値やシステムの使用効率を引用することはまったく無意味である． 評価の実際的な理論がないので，異なった作業負荷に対する同じシステムの性能や，同一の作業負荷に対する異なったシステムの性能を比例計算することは，残念ながら不可能である．実際，いかなる重要な性能理論にもぜひとも必要な要件は，それがそのような比例計算の規則を備えていることである．

この節では，その測定にはハードウェアモニタが最も適切であるような，リソースの使用効率の測定について議論した．マルチプログラミングシステムにおけるソフトウェアモニタの原理についての議論は7-6節を，またタイムシェアリング・システムにおける手法は9-5節をそれぞれ参照されたい．

4-7　多重サーバ・モデル［CH, CO, HO, KA 参照］

基本的な多重サーバ・モデルが2つ存在する(**図 4-7.1** 参照)：(1) 並列サーバと (2) パイプライン・サーバである．基本的な並列サーバ・システムは，同一の能力を持つ m 個のサーバから成る．それぞれのサーバは1つのジョブを完全に処理できるので，m 個までのジョブを同時にサービスすることができる．基本的なパイプライン・サーバ・システムは m 個のサーバから成り，そのおのおのは，ジョブの1部に必要な機能だけに**専門化**されている．どのジョブも m 個のサーバをすべて通って後戻りなしに1度に1ステップずつ進んで行かなければならない．したがって“バケツリレーの行列”の配置ができることになり，いちどきに m 個の異なったジョブを処理することができる．

並列システムと違って，パイプライン・システムでは，各ジョブは処理装置を次々と通って行かねばならない．したがって，たとえすべてのジョブが同時に到着したとしても，その**最初の**ジョブが完了するまでにかかる時間遅れは，単一サーバ・システムでかかる時間よりも長くなることもあり得る．しかし，

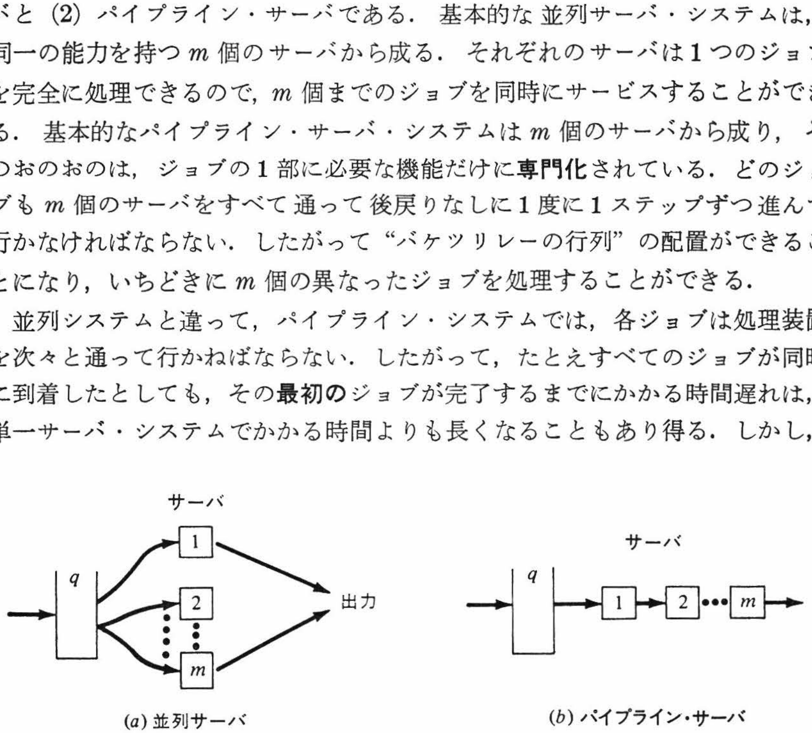


図 4-7.1　多重サーバ・システムの2つの型

パイプラインが満杯になるまでにかかるはじめの時間遅れの後では，パイプラインの中にあるいくつかのジョブは並行処理されるので，ジョブの出力フローは，並列サーバ・システムに匹敵するものとなる．パイプライン処理の思想は，IBM System/360 Mod 91 や CDC-7600 などの非常に高速な中央処理装置(CPU)の設計に使われた思想である．それらのシステムでは，“ジョブ”は1機械命令である；1つのプロセッサは典型的には命令語の読出しを扱い，別の1つがアドレス演算(インデクシング)を，別の1つ(または複数個)が加減算を，さらに他のいくつかが乗除算を取り扱う．

各セグメントのサービス時間が t_i の k 個のセグメントから成るパイプラインによって n 個のジョブを処理するのにかかる時間 T_n に対する簡単な公式は次のようになる．

$$T_n = \left(\sum_{i=1}^{k} t_i\right) + (n-1)\mathrm{Max}(t_i)$$

並列サーバ・システムは，概念的にはもっと簡単で，パイプラインが持つ“満杯にするためと空にするための時間遅れ”はないが，しかし，そのプロセッサは単に専門化された1つの機能ではなくすべての機能を含んでいなければならないので，通常，コストがより高くなる．また，パイプライン・システムには少なからぬ効率の損失がある．なぜなら，もしプロセッサの処理時間が同じでないならば，並びの前の方のプロセッサで長い時間かかると，後の方のプロセッサに遊びが生じるからである．

パイプライン処理と並列処理を結合することは実現可能で，またしばしば得策である．たとえば，CDC-6600 システムのパイプライン化された CPU は，2つの乗算器と2つの増分器を使っている．システム構成のもっと大きなレベルで見ると，CDC-6600 は10個の周辺機器処理装置を含んでおり，これらは

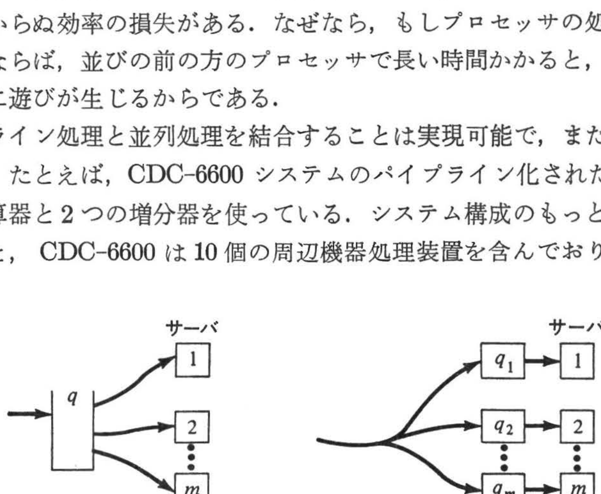


図 4-7.2 並列サーバ・システムに対する待ち行列の2つの型

すべて，入出力処理(および他のタスク)を制御するために並列に作動することができる．

能力がすべて同じの m 個のサーバから成る並列サーバ・システムを考えよう．サーバを，すべての作業負荷が利用できるようにリソースをプールしたもの(**図 4-7.2**)と考えるならば，スケジューリングの柔軟性が最も高くなると思われる．個々のジョブの時間要求は不確定であるから，ジョブがプロセッサを必要とする前にジョブの待ち行列を分解してジョブに確定したプロセッサを割り当てることは，賢明ではないと思われる．この特性は**図 4-7.3** によって確かめられる．この図は，同一のジョブ・ストリームについて，2つの，$m=2$ プロセッサ・システムの性能の比較表を示したものである．1つの場合には，銀

1	2	3	4	5	6	7	8	9	10	←ジョブ
1	4	8	2	6	5	9	2	4	1	←X

(*a*)　実行時間ベクトル

	1	2	3	4	5	
S_1	1	4	8	2	6	X
	1	5	13	15	21	Q

	6	7	8	9	10	
S_2	5	9	2	4	1	
	5	14	16	20	21	Q

性　能

サーバ1

$\bar{t}_e = 55/5 = 11; F = 120/357; U_1 = 1$

サーバ2

$\bar{t}_e = 76/5 = 15.2; F = 45/329; U_2 = 1$

全体としては

$\bar{t}_e = 13.1 \quad F = 0.194$

(*b*)　分離待ち行列システム

1	2	3	4	5	6	7	8	9	10	
1	4	8	2	6	5	9	2	4	1	X
1	4	9	6	12	14	21	16	20	21	Q
1	2	1	2	2	1	2	1	1	1	サーバ

全体としては

$$\bar{t}_e = \frac{124}{10} = 12.4 \qquad F = \frac{10}{47.258} = 0.212$$

(*c*)　単一待ち行列システム

図 4-7.3　分離待ち行列および単一待ち行列をもつ並列処理システムの例

行の出納係窓口やスーパーマーケットの勘定台でよく見られるように，処理を待っているジョブは，2つの分離した待ち行列に分割される．2番目のシステムでは，プロセッサは2つとも，1つの待ち行列からジョブが供給される；各ジョブは，先に空いたプロセッサの方に割り当てられる．2番目のシステムの性能は，平均経過時間および F の値の両方ともについて，1番目のものより良くなることがわかる．単一の待ち行列から多重サーバへジョブを供給する方法の方が優れているにもかかわらず，今日実際に見られる待ち行列の状況では，各サーバに対し別々の待ち行列を持つ方が好まれるように見える．考えられる理由のいくつかは次のものである．

1. 心理的要因，すなわち，待ち行列の監督者が居なければ，人々は長い行列で待つよりも，最も短い行列に殺到する傾向がある．
2. 単一の待ち行列では，ジョブ/顧客をいくつかの小さな空間でなく1つの大きな空間に貯めておくことが必要になる．
3. しばしば，ある種のパイプライン処理を複数の待ち行列配置で行なうことが可能となる．これは，スーパーマーケットの勘定台を見れば明らかで，各勘定台では，1人のレジ係が1人の客の代金勘定をしている間に，次の客が自分のワゴンの中味を勘定台の上に取り出している．もちろんこの例は，各サーバに長さ1の待ち行列を置くということを正当化しているにすぎない；可能な最後の時点で，作業負荷の残りは共通の待ち行列から取られなければならない．

多重サーバ・システムでたびたび生じる別の質問は，処理速度の早い1つのサーバと，処理速度の遅い複数のサーバとの利点の比較である．待ち行列についての考察はさておき，次にあげることが，この選択に関係する．

1. 遅いサーバに比べて速いサーバはコストがかかる．
2. 多重サーバ・システムの方が，サーバの冗長性により信頼性が高くなる(第3章参照)．

コストとスピードの関係を考えてみると，遅いプロセッサから出発して考えるならば，速いプロセッサのスピード増大の度合は，コスト上昇の度合よりずっと大きい場合が多い(少なくとも計算機工学においては)．しかし最終的には，高速化技術の現在の技術水準の限界に近づくにつれて，コスト上昇の度合は，スピード増大の度合より大きくなる．極限に到達したときには，それ以上のスピードの改善は並列処理によって**のみ**可能であり，またその場合でも，シ

ステムの性能が他のリソースのスピードによって制限されているならば，それは満足すべきものではないだろう．

1つの特別な比較の論拠として，単一の速いプロセッサから成るシステム S の性能と，それぞれのスピードが S のプロセッサの $1/m$ である m 個のプロセッサを持つシステム M の性能とを対比してみよう．平均応答時間について見れば，どちらのシステムの方が良いであろうか．システム S は少なくともシステム M と同じ位のスピードがあり，また通常は M よりも速いだろうということが容易にわかる．その理由として次のものがあげられる．

1. 1つのジョブから成るジョブ・ストリームに対しては，明らかにシステム S の方が速い．
2. 同じ長さの k 個のジョブから成るジョブ・ストリームに対しては，$k < m$ であるならば，システム S の方が速い．なぜなら，システム M で t 秒必要であれば，システム S では tk/m 秒必要になるからである．
3. 同じ長さのちょうど m 個のジョブから成るジョブ・ストリームに対しては，スループット，すなわち，m 個のジョブがすべて終了するのに要する時間は，同じになるが，しかしシステム S は，最後のジョブを除いたすべてのジョブ処理をシステム M より早く終了する；両者はともに，最後のジョブ処理を同時に完了する．
4. 長さが同じ任意個数のジョブから成るストリームに対しては，簡単な拡張によって上記の結果が適用される．

ジョブの実行時間が同じでないような，より一般的で実際的な場合には，議論はそれほど簡単ではない．しかし，直観的に，スピードの速い単一サーバ・システムの方に歩がありそうだと思われる．

スピードの速い単一サーバ・システムのもっと大きな強みは，それは**単一の長いジョブ**に対しては自動的に良い性能を与えるということである．多重サーバ・システムによってもそれと匹敵する位のスピードを得るためには，ある人(またはある物)は，多重プロセッサが実行できる形でジョブを表現しなければならない．現在のところは，これは簡単な仕事ではない．将来，プログラミング言語は，ユーザが，配列演算や新しい言語要素をどんどん使うことによって，複雑なプログラムの論理を，コンパイラや他の言語トランスレータが簡単に並列演算を見つけ出すことができるような形で表現することを促すであろう．

最後に，われわれは，並列サーバシステムが持っている重要な潜在的優位点，すなわち，このシステムの方が信頼性が高いということに言及しなければなら

ない．ここで強調しておかなければならないのは，潜在的ということばである；多重サーバによれば可能となる信頼性の改善は，いかなる個数のプロセッサのプールに対しても“割り当てを遅らす”ことに力点をおいたシステム設計によって実際に実現される．信頼性の改善の評価方法については，3章を見られたい．

4-8 ある簡単なジョブ・プロセシング・シミュレータの概要

この章で論じられた原理を使って，われわれはこれから，ジョブ・プロセシング・モデルの簡単なシミュレータの構成とプログラミングについて考えよう．シミュレーションの方法には次の2つの異なった方法がある．

1. GPSS(IBM の *G*eneral *P*urpose *S*imulation *S*ystem)や SIMSCRIPT のようなシミュレーション言語を使って，シミュレータをプログラムする(GO, GP 参照).
2. FORTRAN, APL, PL/I などの汎用プログラミング言語を使って，シミュレータをプログラムする．

特別なシミュレーション言語は，まず言語自身を勉強しなければならないけれども，プログラムするのが簡単である．特に，多くのシミュレーション問題に取りくまねばならないならば，簡単であると言ってもさしつかえなかろう．これらは特別な便宜をいくつか用意しており，ユーザがそれらを使えば，次に掲げる機能の面倒な細かい部分は自動的に行なわれるので，(適切なステートメントによって)何の苦もなくこれらの機能を記述することができる．

1. ユーザが与えた分布関数から，ジョブを記述する数値を生成する．
2. 作業負荷とリソースについての統計を収集し，プリント出力する．
3. ユーザがたやすくシステム構造を規定できるようにする．

これらは非常に便利な強みであるが，この節では，われわれは汎用プログラミング言語に適したシミュレータの概要について述べよう．この方が，われわれを**完全な**シミュレーション・プログラムにより近づけてくれるし，またそれゆえ，シミュレーションの仕事全体の“感じ”を与えてくれる．それはまた，われわれがシミュレータやシミュレーション言語の内部動作をまざまざと思いうかべたり，そのありがたみを理解したりするときに役に立つであろう．汎用プログラミング言語で書かれたシミュレータが専用シミュレータよりも実際上で優れている点のいくつかは，次にあげる点である．

1. 複雑なモデルを規定したり出力レポートの書式を決めたりすることが完全に自由なこと.
2. たいていの場合, データ(たとえば測定データ)を, シミュレータの一部として作られたのではないプログラムと共有することが, ずっと容易なこと.
3. 機械の効率がずっと良いこと.

シミュレーション言語が改良されるにつれて, これらの優位点はあまり言われなくなってきている.

シミュレーション・モデルを記述するのには, われわれはこの章の前の方で論じたカテゴリに従うことにする；すなわち, 作業負荷の記述, リソース(サーバ)の記述, およびシミュレータの論理(スケジューリングと性能尺度を含む)である. 図 4-8.1 はモデルの一般的な構造を示したものである. この詳細についてはこの節の残りで議論される.

作業負荷の各ジョブは3つの数値で記述される(タイプⅡの記述).

1. 到着時刻
2. サービス(実行)時間
3. 準備時間(これはまた後処理時間に等しいと考えられる)

ユーザは, 上記の3つの数値を規定する方法として, 5つの, **作業負荷番号** (workload code)と呼ばれるオプションから1つを選ぶ. それらの選択番号は

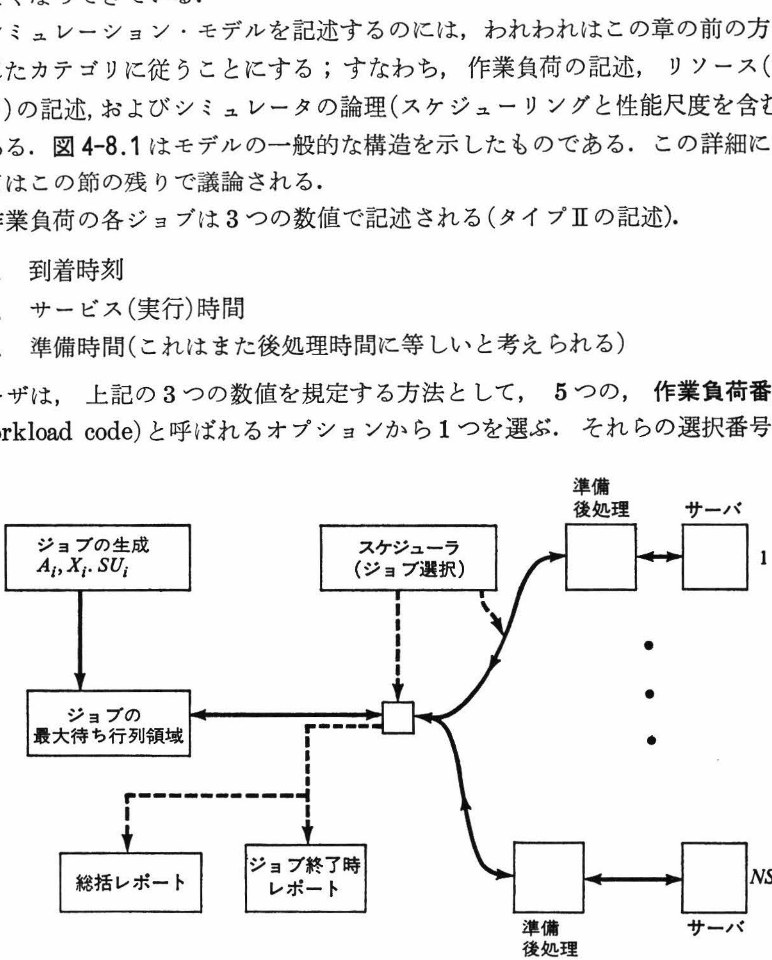


図 4-8.1　一般的な SIMJOB の構成

次のものである.

1. ユーザによって規定される定数値
2. ユーザから直接与えられる明確な数値
3. ユーザによって決められた平均値を持つ指数分布から生成された乱数
4. ユーザから与えられた累積確率の表から生成された乱数
5. 最後のシミュレータ実行時に選んだ作業負荷番号を保持する(連続して実行するときに有用である)

サーバを記述するには, 1つのサーバごとに, 1つの**スピード係数** (speed factor)と呼ばれる数値が必要であり, これはサーバに送られるジョブのサービス時間を縮尺するのに使われる. ユーザはまた, **待ち行列の最大長**を規定する; これを超過すると, それ以上のジョブ発生が一時停止される. もう1つのオプションは, **待ち行列の型**である: (1) ジョブがサーバに送られる時点では, すべてのサーバは単一の待ち行列からジョブをとり出す, (2) 入力してきたジョブは, 到着した時点で, それのサービス全体について, 実効長さが最も短い待ち行列を持ったサーバに割り当てられる. 後者のサーバの選択は, 次のようにして決められる. 各サーバの待ち行列の長さをそれぞれのスピード係数で割り, 待ち行列の実効長さを求める. また, 実効長さが同じ待ち行列が複数個ある場合には, スピード係数が最も大きいサーバを選択する. ユーザによって与えられるもう1つの主要なパラメータは, **スケジューラ番号**であり, これは, スケジューリングは次に掲げるうちのどの方法で行なってほしいかを規定するものである; FCFS(first-come-first-served), SXFS(shortest-job-first), RR (round robin)など. いくつかのよく使われるスケジューラはシミュレータに組み込まれる; 他のスケジューラを付加することもきわめて容易である. なぜなら, スケジューリング変数を最小化することによってスケジューリングが行なわれるように論理が組まれているからである(タイブレーキングのためにFCFS も組み込まれている). スケジューリングが横取り方式であるということもあり得るので, ユーザは**タイム-スライス**のパラメータを供給しなければならない.

シミュレータはもちろんプログラムであり, その論理を理解することは, プログラム解析のよい練習になる. まず最初に, **図 4-8.2** に示すような, 実行の見本(FORTRAN によるシミュレータからとったもの)を見ることが理解に役立つ. 一番初めから読んでいくと, まずシミュレータへの入力データが示され

```
******SIMULATOR  OF  MULTIPLE  JOBS  ON  MULTIPLE  SERVERS******

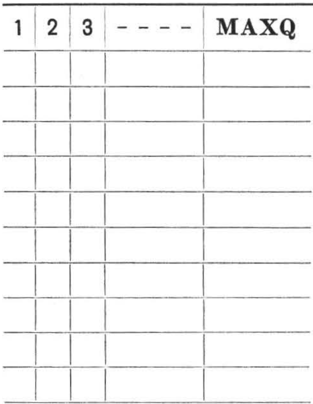
           RUN    1
NO.SERV. NO.JOBS   MAX Q    Q TYPE   QUANT.
   2.0      4.0      4.0      1.0      1.0        .
SCHED      W.L.CODES                SET-UP BIAS PRNT OP.
   4.0      2.0      2.0      1.0      0.0      1.0
SERVER SPEED FACTORS
  1.000   2.000
ARRIVAL TIMES(OR MEAN OR TABLE SIZE)
  0.0      0.500   1.000   1.500
SERVICE TIMES(OR MEAN OR TABLE SIZE)
  2.000   3.000   2.000   1.000
JOB     SET-UP TIMES
  0.100   0.0      0.0      0.0
********TRACE OF SERVER START/END EVENTS********
TRACE CODES
11=SETUP START, 12=SETUP END
21=SERVICE START, 22=SERVICE END
31=SETDWN START, 32=SETDWN END, *****=JCB END
   CLOCK      JOB   SERVER CODE SVCE.T. COMP.T. ARR.T.
    0.0        1       2   11    2.00    0.0     0.0
    0.10       1       2   12    2.00    0.0     0.0
    0.10       1       2   21    2.00    0.0     0.0
    0.50       2       1   11    3.00    0.0     0.50
    0.60       2       1   12    3.00    0.0     0.50
    0.60       2       1   21    3.00    0.0     0.50
    1.10       1       2   22    2.00    2.00    0.0
    1.10       1       2   31    2.00    2.00    0.0
    1.20       1       2*****    2.00    2.00    0.0
    1.20       3       2   11    2.00    0.0     1.00
    1.30       3       2   12    2.00    0.0     1.00
    1.30       3       2   21    2.00    0.0     1.00
    1.60       2       1   22    3.00    1.00    0.50
    1.60       2       1   31    3.00    1.00    0.50
    1.70       2       1   32    3.00    1.00    0.50
    1.70       4       1   11    1.00    0.0     1.50
    1.80       4       1   12    1.00    0.0     1.50
    1.80       4       1   21    1.00    0.0     1.50
    2.30       3       2   22    2.00    2.00    1.00
    2.30       3       2   31    2.00    2.00    1.00
    2.40       3       2*****    2.00    2.00    1.00
    2.40       2       2   11    3.00    1.00    0.50
    2.50       2       2   12    3.00    1.00    0.50
    2.50       2       2   21    3.00    1.00    0.50
    2.80       4       1   22    1.00    1.00    1.50
    2.80       4       1   31    1.00    1.00    1.50
    2.90       4       1*****    1.00    1.00    1.50
    3.50       2       2   22    3.00    3.00    0.50
    3.50       2       2   31    3.00    3.00    0.50
    3.60       2       2*****    3.00    3.00    0.50
********** SUMMARY  STATISTICS **********
NO. JOBS    TOT.TIME JOBS/SEC AVE.EXEC. AVE.STR.FAC.
     4       3.600    1.111     2.000         0.933
AVE. ELAPSED  AVE. WAIT P.C. UT.    P.C. S.U./UT.  Q-AVE.      Q OVFLO TIME
    1.775       0.650    83.333      16.667            1.972        0.0

SERVER UTILIZATIONS
SERVER SPEED F.  P.C. UTILIZED
    2  2.0000    100.0000
    1  1.0000     66.6666
     DISTRIBUTION FUNCTION OF STRETCH FACTORS
UNDER 1    1    2    3    4    5    6    7    8    9   10   11   12   13   14
   0.50 0.50 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
   0.50 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
```

図 4-8.2 SIMJOB シミュレータの実行例

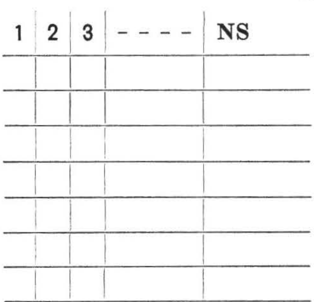
ていてこれは次のものから成っていることがわかる.

1. サーバの個数
2. シミュレートされるべきジョブの個数(さもなければ, シミュレートされるべき総時間)
3. 待ち行列に入ることが許されるジョブの最大数(MAX Q); Q タイプ
4. タイム・スライス
5. スケジューラ番号; 1 = FCFS, 2 = SXFS, 3 = LCFS, 4 = RR
6. 到着時刻, サービス時間, 準備時間に対する作業負荷番号
7. プリント出力(トレース)オプション: 1 = どのリソースの状態変化もすべて完全にトレースする, 2 = それぞれのジョブの完了毎にプリント出力する, 3 = すべてのジョブが完了した後での“総括のみ”



記号	意　味
JB	ジョブ番号
A	到着時刻
X	サービス(実行)時間
SU	準備時間(後処理時間)
XL	サービス時間(完了ずみのジョブについて)
TLS	最新の量子(タイム・スライス)時刻
IP	ジョブが待ち行列を作っているサーバ
Q	ジョブが実際に完了した時刻
JS	ジョブの状態
SV	スケジューリング変数

(a) ジョブに関する情報



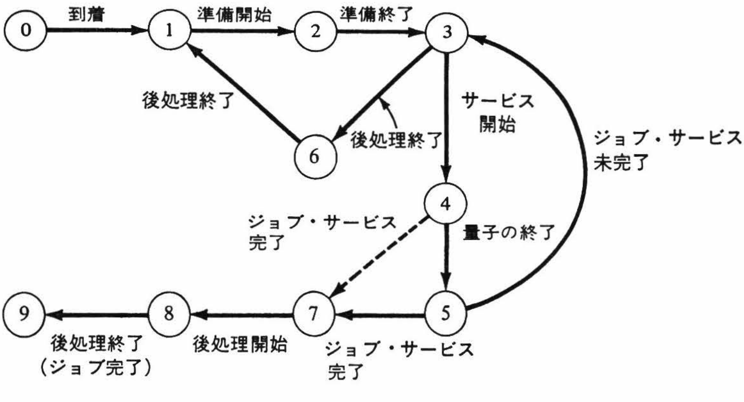
記号	意　味
SF	サーバのスピード係数
RS	サーバの状態
JX	現在サービス中のジョブ
IQL	サーバの待ち行列に並んでいるジョブの数
TST	現在のイベントが開始された時刻
EJ	現在のイベントの終了予定時刻
UT	サーバの使用時間

(b) サーバに関する情報

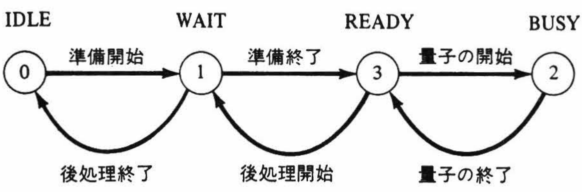
図 4-8.3 シミュレータ・プログラムのベクトル

8. サーバのスピード係数(第2行)
9. 到着時刻または，到着分布表のサイズおよびそれに続いて分布表(作業負荷番号が4または5の場合)
10. 項番9と同じ形式で与えられる，サービス(実行)時間と準備時間

シミュレータの主なデータ構造(**図4-8.3**)は，おのおののジョブの状態を表わしているベクトルの集合および，おのおの**サーバ**の状態を表わしているベクトル集合である．各ベクトルの簡単な定義が図に示されているので，ここでは，それらについて詳しく述べない．それぞれの集合内の1つのベクトルは，**状態ベクトル**(state vector)と呼ばれる(ジョブに対してはJS，サーバに対してはRS)．シミュレータの論理の核心は，どのように状態の値が変化するかである．**図4-8.4**および**図4-8.5**は，状態推移図とその表による表現を示したも



(*a*) ジョブの状態推移



(*b*) サーバの状態推移

図 4-8.4　ジョブとサーバの状態図

行	現在の状態		次の状態		分岐	推移の意味
	サーバ	ジョブ	サーバ	ジョブ		
1	1	2	3	3	1	準備完了
2	1	6	0	1	2	後処理終了；ジョブが未終了
*3	2	4	3	5	3	サーバの量子の終了
4	1	8	0	9	4	後処理終了；ジョブが終了
*5	3	5	3	7	5	サーバの量子の終了；サービス終了
*6	3	5	3	3	5	サーバの量子の終了；ジョブは未終了

(*a*) サーバ-終了表

行	現在の状態		次の状態		分岐	推移の意味
	サーバ	ジョブ	サーバ	ジョブ		
1	0	1	1	2	1	準備開始
*2	3	3	2	4	2	サーバ開始
*3	3	3	1	6	3	後処理開始(ジョブが未終了)
4	3	7	1	8	4	後処理開始(ジョブが終了)

(*b*) サーバ-開始表

*サーバ終了表の行3で，次に行5へ進むか行6へ進むかはプログラムテストにより決定され，行6へ進んだ場合には循環する．同様に，サーバ-開始表の行2, 3のどちらに進むかはプログラム・テストにより決定される．

図 4-8.5 図4-8.4の状態図に対するサーバ-開始表とサーバ-終了表

のである．この種の図は，シミュレータを計画したり，設計したり，プログラムしたりするのに非常に貴重である．

図4-8.6には，かなり大ざっぱなフローチャートの形で，シミュレータ・プログラムの論理が示されている．フローチャート上の番号のついた箱はそれぞれ，次のように記述される．

1. 入力パラメータがすべて受取られ，マスター・クロックCMを含む変数の値が初期化される．
2. 待ち行列が一杯でないなら，それを満たすためにジョブを生成する．新しいジョブの到着時刻がCMの値より早い場合が見つかると，これは待ち行列のあふれとして記録される(このための論理は示されていない).
3. 次の到着時刻と次のサーバ終了時刻のうちの早い方を次のイベント(事象)として探す．
4. 次の到着時刻 ≤ 次のサーバ終了時刻であれば，CMは，次の到着時刻

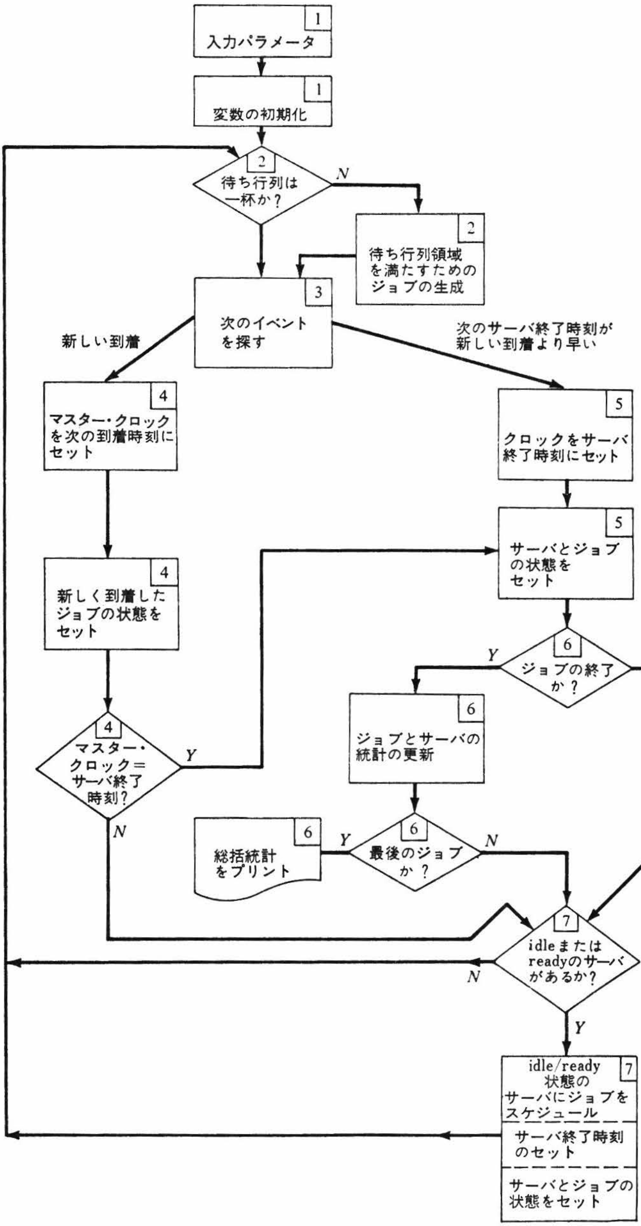


図 4-8.6　SIMJOB の基本論理

にセットされ，したがって新しく到着したジョブの状態は1にセットされる．CMの現在値が次のサーバ終了時刻に等しく**ない**ならば，7の箱へ飛ぶ．そうでなければ，次へ進む．

5. 項番3の結果がサーバ終了時刻であれば，CMはこの時刻にセットされ，ジョブとサーバの状態は，「サーバの終了」とセットされる．
6. 項番5でサーバ内のジョブのサービスがまだ完了していないならば，プログラムは，サーバ-ジョブ・スケジューリングへ飛ぶ．ジョブが完了していれば，ジョブの統計が更新され，またこれが最後のジョブならば，統計の結果が計算され，プリント出力される．そうでないなら，プログラムは，サーバ-ジョブ・スケジューリングへ飛ぶ(7の箱)．
7. サーバ-ジョブ・スケジューリング：IDLEかあるいはREADY状態のサーバがないならば，項番2へ戻る．READY状態の1つのサーバが完了ずみのジョブを持っていれば，そのサーバに対して後処理を開始する．他のREADYまたはIDLEのサーバのそれぞれに対して，規定されたスケジューラを使ってそのサーバがサービスすべき次のジョブを決定する．もしそのジョブがそのサーバ内にあれば，サーバをスタートさせる．サーバがIDLEであれば，選択されたジョブの準備を開始する．サーバはREADYであるが，しかしそれが次にサービスすべきジョブを含んでいないならば，(次のジョブのための準備を見越して)，自分のジョブに対する後処理を開始する．

演　習　問　題

1. ジョブ・プロセシングのシミュレーション・モデルは数学的モデルより一般的であり得るのはなぜかを説明せよ．解析的(数学的)モデルがシミュレーション・モデルより優れている点は何か？
2. ジョブ・プロセシング・モデルにおける“ジョブ”という用語を定義し，説明せよ．
3. 読者自身のことばで，ジョブ・ストリームの性能とリソースの性能との間の違いを説明せよ．
4. テスト用に，あるいはシミュレートされる作業負荷のためにジョブを選ぶ際に考えねばならない一般的な基準(少くとも3つ)は何か？
5. 合成作業負荷とは何か？　選択されたジョブによる作業負荷よりも優れ

ている点は何か？　また不利益は何か？

6. 多くの大学で見られるような伝統的な形の計算機センターを考えよう．ジョブは特別な窓口で受付けられ，ユーザの名前により，アルファベット順に並んだ置き場に返却される．

(a) このシステムのジョブ・ストリームの性能に影響を与える要因のうち，機械-ソフトウェア以外のものを少くとも3つあげよ．

(b) ユーザが自分のカードを読取機に置き，出力をプリンタから受け取ることができるように，ジョブ受付け窓口にカード読取機とプリンタを設置することを考えよう．このことは性能にどのように影響を与えそうであるかを論じ，その理由も述べよ．

7. すべてのジョブが同時に到着する（たとえば同時に1つのバッチにまとめられる）ジョブ・ストリームが与えられている．タイブレーキングのためには任意の選択を行なうFCFSスケジューリングを仮定する．

$$X = 10, 1, 4, 2, 1, 5$$

(a) スループットを計算せよ．

(b) 平均経過時間を計算せよ．

(c) F(短いジョブのフィギャ・オブ・メリット)の値を計算せよ．

(d) Xの値の増加順になるようにXを並べかえて，(a)から(c)までの問に答えよ．

(e) Xの値の減少順になるようにXを並べかえて(a)から(c)までの問に筋えよ．

(f) (d)と(e)の結果を含んだ表を用意し，各性能パラメータを比較して論評せよ．

8. 準備時間と後処理時間はすべて等しい(＝0.2)次のようなタイプIIの作業負荷が与えられている．

A	0	0	0	0	0
X	8	1	4	2	1
S	0.2	0.2	0.2	0.2	0.2

(a) FCFSスケジューリングの場合について，平均経過時間，スループット，F，準備/後処理に費された総時間の割合を計算せよ．

（b） タイム・スライスが1.0のラウンド・ロビン・スケジューラの場合について，問(a)の性能パラメータを計算せよ．

（c） 各パラメータについて(a)と(b)の結果を比較せよ．

9. 単一サーバ・システムの平均待ち行列長は次のように定義される．

$$\overline{Q} = \frac{\text{ジョブの総(待ち + 処理)時間}}{\text{ジョブ・ストリームの経過時間}}$$

平均待ち行列長と，平均経過時間およびスループットとの間には次のような関係があることを示せ．

$$\overline{Q} = \bar{t}_e H$$

10. 1台のCPUと2台のチャネルを持つシステムを100時間の間ハードウェア・モニタでモニタしたところ，それぞれの使用時間はCPUが60時間，チャネルAが20時間，チャネルBが30時間，オーバラップしないCPU時間は10時間，両方のチャネルがCPUとオーバラップする時間は8時間，遊び時間は20時間であることがわかった．

（a） CPUが“待ち状態”(“チャネルの終了”を待っている)であった時間はいくらか？

（b） CPUがチャネル1台だけとオーバラップしていた時間はいくらか？

11. セグメントがk個でi番目のセグメントの処理時間がt_iであるパイプライン・システムに対する次のような一般的な結果を導出せよ．

（a） 1つのジョブを処理するのに要する時間

$$T_1 = \sum_{i=1}^{k} t_i$$

（b） n個のジョブをパイプラインを通して処理する時間

$$T_n = \left(\sum_{i=1}^{k} t_i\right) + (n-1)\,\mathrm{Max}\,(t_i)$$

（c） 厳密に逐次的な非パイプライン処理に対するパイプライン処理のスピード改善率

$$I = \frac{n\sum_{i=1}^{k} t_i}{\left(\sum_{i=1}^{k} t_i\right) + (n-1)\mathrm{Max}(t_i)}$$

（d）　もしすべてのパイプ・セグメントが必要とする処理時間が等しいならば，上の式は次のようになる．

$$I = \frac{nk}{n+k-1}$$

12．　スピード係数が1，タイム・スライスが2のサーバが1台でジョブ・ストリームが

A	0	0	0	0	0	0	0	0	0	0
X	1	3	8	2	5	1	3	9	2	4
SU	0	0	0	0	0	0	0	0	0	0

であるとしてSIMJOB SIMULATORを使い，以下の問に答えよ．

（a）　FCFS, RR, SXFSスケジューラについて，スループット，平均引き伸し率，準備時間を比較せよ．

（b）　RRとFCFSの性能の結果はどのように説明できるか？

（c）　3つのスケジューラに対して，X_iをどのように並べ変えれば性能を最もよく改善できるか？　どのように並べ変えれば性能が最悪となるか？

（d）　X_iの並べ変えによる影響が最も少ないのはどのスケジューラか？　どのスケジューラが最もよく改善されるか？　またその理由は？

13．　各ジョブの準備時間が0.5であるとして問題12に答えよ．準備時間が各スケジューラの性能に与える影響を互いに比較して論評せよ．

14．　問題12のジョブ・ストリームとスケジューラを使い，それぞれのスピード係数が0.5である2つの並列サーバでストリームを処理する．

（a）　問題12と比べ，性能はどのように異なるか？

（b）　なぜ異なるのか？

15．　各ジョブの準備時間を今度は0.5秒と仮定して問題14に答えよ．

参 考 文 献

1. [AR] Arbuckle, R. A.: Computer Analysis and Thruput Evaluation, *Computers and Automation*, January, 1966.
2. [BU1] Bucholz, W.: A Selected Bibliography on Computer System Performance Evaluation, *IEEE Computer Group News*, vol., 2 no. 8, March, 1969.
3. [BU2] Bucholz, W.: A Synthetic Job for Measuring System Performance, *IBM Systems Journal*, vol. 8, no. 4, pp. 309-318, 1969.
4. [CH] Chen, T. C.: Parallelism, Pipelining, and Computer Efficiency, *Computer Design*, January, 1971.
5. [CO] Conti, C.: System Aspect: System/360 Model 92, *AFIPS Conf. Proc.*, vol. 26, part 11 (1964 FJCC), pp. 81-95.
6. [DR] Drummond, M. E.: "Evaluation and Measurement Techniques for Digital Computer Systems," Prentice-Hall, Inc.; Englewood Cliffs, N. J., 1973.
7. [GO] Gould, R. L.: GPSS/360 An Improved General-Purpose Simulator, *IBM Systems Journal*, vol. 8, no. 1, 1969.
8. [GP] GPSS Manuals from IBM: Application Description (H20-0186); Introductory User's Manual (H20-0304); User's Manual (H20-0326). *IBM Corp.*, White Plains, N.Y.
9. [HE1] Hellerman, H.: Complementary Replacement–A Meta Scheduling Principle, *Proc. Second ACM Symp. on Operating Systems Principles*, Princeton, N.J., Oct. 20-22, 1969, pp. 43-46.
10. [HER] Herman, D. J., and F. C. Ihrer: The Use of a Computer to Evaluate Computers, *AFIPS Conf. Proc.*, vol. 25 (1964 SJCC), pp. 383-395.
11. [HO] Holland, F. C., and R. A. Merikallio: Simulation of a Multiprocessing System Using GPSS, *IEEE Trans. Syst. Sci. Cyb.*, vol. SSC-4, no. 4, pp. 395-400, November, 1968.
12. [HU] Hutchinson, G. K., and J. N. Maguire: Computer Systems Design and Analysis Through Simulation, *AFIPS Conf. Proc.*, vol. 27, part 1 (1965 FJCC), pp. 161-167.
13. [IH] Ihrer, F. C.: Computer Performance Projected Through Simulation, *Comput. Autom.*, vol. 17, no. 4, pp. 22-27, April, 1967.
14. [KA] Katz, J. H.: Simulation of a Multiprocessor Computer System, *AFIPS Conf. Proc.*, vol. 28 (1966 SJCC), pp. 127-139.
15. [KE] Keefe, D. D.: Hierarchical Control Programs for Systems Evaluation, *IBM Systems Journal*, vol. 7, no. 2, pp. 123-133, 1968.
16. [LE] Lehman, M. M., and J. L. Rosenfeld: Performance of a Simulated Multiprogramming System, *AFIPS Conf. Proc.*, vol. 33, part 11 (1968 FJCC), pp. 1431-1442.
17. [MU] Murray, R. J.: The logic and Usage Recorder, *AFIPS Conf. Proc.*, vol. 35, pp. 219-229, 1969.
18. [PA] Patrick, R. L.: Measuring Performance, *Datamation*, vol. 10, no. 7, pp. 24-27, July, 1964.
19. [RO] Roek, D. J., and W. C. Emerson: A Hardware Instrumentation Approach to Evaluation of a Large System, *ACM Proceedings of 24th National Conf.*, pp. 351-367, 1969.
20. [SC] Schulman, F. D.: Hardware Measurement Device for IBM System/360 Timesharing Evaluation, *Proc. 22nd ACM Natl. Conf.*, pp. 103-109, August, 1967.
21. [SM] Smith, E. C.: A Directly Coupled Multiprocessing System, *IBM Systems Journal*, vol. 2, no. 3, pp. 218-229, September-December, 1962.
22. [SR] Sreenivasan, K., and A. J. Kleiman: On the Construction of a Representative Synthetic Workload, *Comm. ACM*, vol. 17, no. 3, p. 127, March, 1974.
23. [WA] Walter, E. S., and V. L. Wallace: Further Analysis of a Computing Center Environment, *Comm. ACM*, vol. 10, no. 5 pp. 266-272, May, 1967.

第5章

待ち行列理論

待ち行列(queue, waiting line, congestion)理論は，多数の客が少数の窓口リソースの処理を要求しているような現実のシステムに似たある抽象モデルを記述し解くことに関する理論である．本章で紹介される解析的待ち行列理論は，輻輳問題を取扱う際に技量，直観，洞察を与えるのに役立ち，もちろんモデルが現実によくあてはまる場合には，(パラメータの変化の影響を正確に容易に評価でき)解析解法の恩恵を得ることができる．しかし第4章で述べたように，解析モデルは比較的少数の現実世界のシステムにしか適用できず，非形式的な方法，シミュレーション法などの方が実際にはより多く用いられている．

歴史的には，待ち行列理論は1909年デンマーク人技術者 A. K. Erlang が電話システムの輻輳の数学的解析についての，今では古典的となった論文を出したときから始まった．Erlang が取り組んだ問題は，簡単に言えば，接続されている電話加入者数と，電話をかける確率と，電話をかけている時間の長さの分布と，電話会社によって設置される“中継回線”の数の間の関係である．

回線は次のように理解されるだろう．電話の非常に初期には，各加入者と他の加入者を結ぶために別々の線が用いられていた．それで n 人の加入者に対して，$n(n-1)/2$ 本の線が必要であった．n が大きくなるにしたがい，各加入者が同時に電話することはないので，このような多数の線は必要でないことがすぐに認識された．すべての加入者(利用者)がつなぐことのできる**回線**と呼ばれる少しの線を供給するだけでよい．k を設置された回線の数とすれば，n 人の利用者の中の(最大)k 人が同時に電話することができる．いつでも k 個の通話がされていたら，通話しようとしている他の利用者は回線を得ることができず，接続できない．(その利用者は相手が話し中の信号を受け取る．)定性的には，そのようなシステムの設計の結果は容易に理解される．もし k が小さすぎると利用者が通話できない可能性が高く，利用者の不満が増し，また，完

了された通話にのみ課金されるので，収入が減ることとなる．逆に k が大きすぎると，大きな設備投資が必要とされ，その多くが大部分の時間稼動していない可能性が大である．

利用者のそのシステムに対する要求のある現実的な表現を基に，k を決定する合理的な根拠が重要である．この種の問題は今日では，多数の利用者の通信，計算機システムの設計に(さまざまに形をかえて)残っている．Erlang は単に電話回線問題のすっきりした理論を与えただけでなく，その中で利用者とシステムの振舞のかなり一般的な確率的表現法にいくつかの寄与をした．これらのいくつかについて本章で学ぶ．

第4章で議論されたように，各モデルはその作業負荷の表現をしなければならない．次節では，ジョブ到着時刻の最も重要なモデル，Poisson 到着過程を述べることから始める．次に物理学における保存則と同じような役割をする平衡に関する重要な差分微分方程式を導き出そう．この方程式(というよりはむしろ方程式の系)の平衡状態の解が次に導かれ，並列窓口システム，タイムシェアリング・システム・モデルを含む2, 3の重要で特別な場合への適用を行なう．

5-1 Poisson 到着(発生)過程

この節では待ち行列の1つの側面である客の要求の到着を，そのシステムの他のものから分離する．サービスを要求できる n 人の客の集団(n は大きいとする)を考えよう．しかしここではサービスの終了については考えない．客の到着の様子を表わすと考えられるいくつかのモデルの中で，特に，Poisson(あるいは指数分布）と呼ばれるものが解析される．Poisson 到着機構は2つの理由で選ばれた．すなわち (1) それが常にではないが，しばしば実際の到着の模様に合致している．(2) それが，待ち行列モデルで解析的に解ける数少ないものの1つである．

数学的な待ち行列理論はその変数を表現するためにいくつかの記号を必要とする．図 5-1.1 は定義や，変数間の簡単な関係を参照するのに便利なように用意されたものである．記号と記法は著者の間で統一されておらず，2人の著者が同じ記号を別の意味に使うことは珍しいことでないことに読者は注意されたい．

微少時間間隔 h の間におのおのの客が要求を出す(到着する)確率は ph で同じであるという意味において，すべての客 n は統計的に皆同じであると仮定しよう．客の要求はすべて独立であるから，n 人の客のうちのだれか(特定の1

記号	意　味
h	微小時間間隔
j	待ち行列長(処理中のジョブも含める)
j	平均待ち行列長(j の平均値)
$\bar{k}$	処理中のジョブを除いた平均待ち行列長
n	システム内の客の総数($= j$ の値の考えうる最大値)
ph	ある1人の客が時間 h の間に要求を出す確率
$P_j(t)$	システムが時刻 t で状態 j である確率：状態 j とは待ち行列に(処理中のジョブを含めて) j 人の客がいることを示す.
r	$\alpha/\omega =$ トラフィック密度(単位アーラン)
T	平均“思考”時間
$\bar{t}_q = \bar{\omega}$	平均行列待ち時間(処理時間も含める)
$\bar{t}_S = 1/\omega = C$	平均処理時間
$\bar{t}_\omega$	平均待ち時間(処理時間を除いた，待ち行列にいる時間の平均)
$\alpha_j h$	システムが状態 j にあるとき客が待ち行列にやってきて並ぶ確率
λ	$= pn$(離散型 Poisson 分布の平均)
ρ	利用率($0 \leqslant \rho \leqslant 1$) Poisson・システムでは $r = \rho$
$\omega_j h$	システムが状態 j にあるとき，客が処理を終え待ち行列から離れる確率
$\bar{\alpha}$, $\bar{\omega}$	j について $\bar{\alpha}_j$, $\bar{\omega}_j$ の平均
σ_S	処理時間の標準偏差

図 5-1.1 待ち行列理論における記法

人ではなく)が要求を出す，すなわち微小時間間隔 h の間に待ち行列に到着する確率は $nph = \alpha h$ である．ここで α は pn の省略形である．また微小時間間隔 h の間に2人以上の客が到着する確率は αh よりずっと小さいと仮定しよう．以下の定義が必要となる．

定　義　**待ち行列の客の数がちょうど j であるとき，システムは状態 j にあると言われる．$P_j(t)$ は時刻 t でシステムが状態 j にある確率である．ここで任意の t について**

$$\sum_{j=0}^{n} P_j(t) = 1$$

である．

さて時刻 t から微小時量後の $t + h$ までの間のシステムの状態変化について考えよう．特に，時刻 $t + h$ に状態 j に“到達”できるのは，次の場合だけで

ある.

1. 時刻 t に状態 $j-1$ にあり, その後1人の客が待ち行列に到着する場合.

$$確率 = \alpha h P_{j-1}(t)$$

2. 時刻 t に状態 j にあり, その後1人の客も待ち行列に到着しない場合.

$$確率 = (1-\alpha h)P_j(t)$$

時刻 $t+h$ でシステムが状態 j にある確率は上の2つの確率の和となる.

$$P_j(t+h) = \alpha h P_{j-1}(t) + (1-\alpha h)P_j(t) \tag{1}$$

両辺から $P_j(t)$を引き, h で割り, $h \to 0$ の極限をとれば, 式(1)は t の微分方程式, j の差分方程式となる.

$$P_j'(t) = -\alpha P_j(t) + \alpha P_{j-1}(t) \tag{2}$$

ラプラス変換によって解けば

$$sP_j - P_j(0) = -\alpha P_j + \alpha P_{j-1}$$
$$P_j(s) = \frac{P_j(0)}{s+\alpha} + \frac{\alpha P_{j-1}}{s+\alpha} \tag{3}$$

となる. j の値ごとに1つの式が得られる. この連立方程式を解くために, $j=0$ の場合から始める. P_{j-1} は0である. 式(3)は

$$P_0 = \frac{P_0(0)}{s+\alpha}$$

となり, 式(3)に代入して

$$P_1 = \frac{P_1(0)}{s+\alpha} + \frac{\alpha P_0}{s+\alpha}$$
$$= \frac{P_1(0)}{s+\alpha} + \frac{\alpha P_0(0)}{(s+\alpha)^2}$$

となる. 一般に

$$P_j(s) = \sum_{i=0}^{j} P_{j-i}(0)\frac{\alpha^i}{(s+\alpha)^{i+1}} \tag{4}$$

となる. 逆ラプラス変換により

$$P_j(t) = \sum_{i=0}^{j} P_{j-i}(0)\frac{(\alpha t)^i}{i!}e^{-\alpha t} \tag{5}$$

が得られる．

例：　待ち行列に**丁度**3人の客($j=3$)が並んでいる確率は次式で与えられる．

$$P_3(t) = P_3(0)e^{-\alpha t} + P_2(0)(\alpha t)e^{-\alpha t} + P_1(0)\frac{(\alpha t)^2}{2!}e^{-\alpha t} + P_0(0)\frac{(\alpha t)^3}{3!}e^{-\alpha t}$$

さて式(5)の Poisson 到着関数の基本的な性質を調べよう．待ち行列が，たとえば k 人の客だけ**長くなる**確率は待ち行列の初めの長さ j には独立である．これを示すために，時刻 $t=0$ で行列には j の客がいたとしよう．ゆえに $P_j(0)=1$，$P_i(0)=0 (i \neq j)$ となる．$t=0$ からある時間 t 後に，待ち行列は客が k 人だけ増え，長さが $j+k$ になったとする．これらの関係から式(5)は

$$P_{j+k}(t) = \sum_{i=0}^{j+k} P_{j+k-i}(0)\frac{(\alpha t)^i}{i!}e^{-\alpha t} \tag{6}$$

$$P_{j+k}(t) = \frac{(\alpha t)^k}{k!}e^{-\alpha t} \tag{7}$$

となる．ここで後の式は $j=i$ 以外では $P_i(0)=0$ であることから得られた．この式は待ち行列が j から k だけ伸びる確率は j に独立であることを示している．このことは Poisson 過程が時々，ランダム(完全ランダム)とか**無記憶**とか呼ばれる理由である．式(7)を

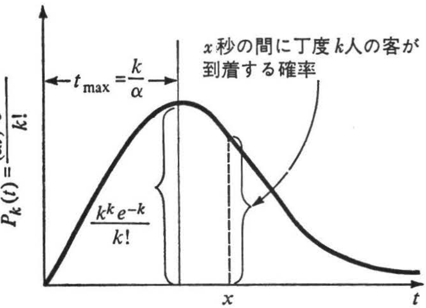
$$P_k(t) = \frac{(\alpha t)^k}{k!}e^{-\alpha t} \tag{8}$$

と書き換えることは有益な簡略化である．この関数は次のことを示している．すなわち，待ち行列が**丁度** k だけ長くなる確率は，小さな t に対しては，k 人の新たな客が到着するのに充分な時間がないので小さくなる．また大きな t に対しては，待ち行列が k 人の客よりももっと長く伸びる確率が高いので，$P_k(t)$ は小さくなる．(図5-1.2参照)

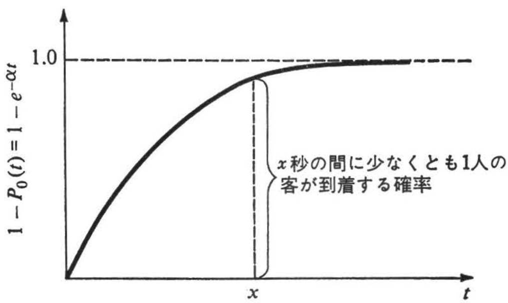
実際には，興味の大部分は**到着時間間隔**の分布，すなわち，時間 t の間に**少なくとも1人**の新たな客が待ち行列に加わる確率である．少しばかり(しかしうまく)記号を変更し，x が経過時刻の変数で t が x のある特定な値とする．

$P_0(x \leq t)$ ＝ 時間間隔 t の間に待ち行列に0人の客が加わる確率

$1 - P_0(x \leq t)$ ＝ 時間間隔 t の間に待ち行列に少なくとも1人の新たな客が加わる確率



(a) ポアッソン分布 (単一kに対する)



(b) 到着時間間隔の指数分布

図 5-1.2 Poisson 到着過程の2つの記述子

式(8)で $k=0$ として

$$P(k \geqslant 1)=P(t)=1-e^{-\alpha t} \tag{9}$$

が得られる．これは**指数分布**であり，Poisson 到着過程の到着時間間隔分布を表わしていると考えられる．到着時間間隔の分布の平均と分散は

$$\text{平均到着時間間隔} = \int_0^\infty t\frac{d}{dt}(1-e^{-\alpha t})dt = \int_0^\infty (\alpha t)e^{-\alpha t}dt$$

$$= \frac{1}{\alpha} \tag{10 a}$$

$$\text{到着時間間隔の分散} = \int_0^\infty t^2 e^{-\alpha t}dt$$

$$= \frac{1}{\alpha^2} \tag{10 b}$$

となる．時刻 t における待ち行列の平均長は

$$\bar{j}(t) = \sum_{j=0}^{n} jP_j(t) = \alpha t \tag{11}$$

である．

式(8)の Poison 分布に対するもう1つの洞察は，離散型 Bernoulli 試行(第3章参照)，すなわち裏か表てか出る面がかたよりがちなコインを n 回投げ上げることで表わされるような連続試行を考えることで得られる．コインの片面を“成功”と呼び，成功する確率はどの投げ上げでも p とする．n 回のうち j 回成功する確率は次式で与えられる．

$$P(j;\ n) = p^j(1-p)^{n-j}\frac{n!}{j!(n-j)!}$$

以下の条件のもとに

$$p \ll 1$$
$$n \gg 1$$
$$\lambda = pn$$

上式の2項分布は Poisson 分布でかなり近似される.

$$P(j;\ \lambda) = \frac{\lambda !}{j!}e^{-\lambda} \tag{12}$$

式(12)は式(8)と形が似ている. 容易に比較できるように再掲する.

$$P_k(t) = \frac{(\alpha t)^k}{k!}e^{-\alpha t} \tag{8}$$

(連続時間モデルで得られた)(8)式と(離散系列モデルで得られた)(12)式の関係をもっと詳しく調べるために, 最初のモデルの連続時間 t を等しい時間間隔 h で細分化し

$$t_n = nh$$

とする. (8)式と(12)式の変数を比較し, 前述の p, n に対する極限条件を用いると

$$\lambda = pn = \alpha t = \alpha nh$$

あるいは

$$p = \alpha h$$

となる. このように Poisson 到着過程はベルヌイ試行系列から生ずるものと考えられ, 各試行は微小時間間隔 h で行なわれ, "成功" は待ち行列に客が到着することを意味する.

要約すれば, Poisson 到着過程は次のような理由でその到着時間間隔が指数分布となる.

1. 処理要求を出す(到着する)独立な客の数が大きい.
2. 任意の微小時間間隔 h で, 1人の客が到着する確率は小さく, その確率はすべての客について等しい.
3. 微小時間間隔 h の間に客が到着しないかあるいは1人到着する確率は,

h の間に2人以上到着する確率よりずっと大きい.

到着時間間隔が指数的に分布している Poisson 到着過程は, もちろんある特殊なクラスの到着過程であり到着時間間隔の分布であるにすぎない. 到着時間間隔の一般的概念を極めて具体的なものとするために, バッチ・コンピュータ・センタへの仕事の到着回数のデータを集めるといった簡単な試行を考えてみよう. 仕事の単位は**ジョブ**と呼ばれ, 物理的には穿孔されたカードのデックであり, 誰かがコンピュータ・センタの"入力"窓口に運んでくる. われわれがこの過程の監視係であり, 次のように仕事を進めるとしよう.

1. 各ジョブが窓口に到着した時刻を記録する. これらの時刻を順番に集めたものをベクトル A と呼ぶ. A の中に n 個の値があるとする.
2. A の差分ベクトル D を次のように計算する.

$$D_1 = A_1$$
$$D_{i+1} = A_{i+1} - A_i \qquad i = 1, 2, \cdots, n-1$$

D_2 から始まるベクトル D は到着時間間隔のベクトルである. その直前のジョブの到着時刻からおのおのの到着時刻を測ることによって, D を直接得ることもできることに注意しよう.

3. さて, 第2章で述べられた手法を用い, D の値の度数分布を作る. すなわち, ある時間間隔 h の時間バケットを選び, そのバケットに 0, h, $2h$ 等とラベルを付ける. そしてバケットの度数を増しながら D の各要素をバケットに入れる.
4. 相対累積度数分布を計算する.

$$c_i = \frac{\sum_{j=1}^{i} f_j}{\sum_{j=1}^{m} f_j}$$

もし到着過程が Poisson であれば, 上で試験的に得られた c_i の値は次式にぴったりと合うはずである.

$$1 - e^{-t'/T}$$

ここで T は**平均**到着時間間隔であり,

$$T = \frac{\sum_{j=1}^{m} t_j' f_j}{\sum_{j=1}^{m} f_j}$$

で得られ，$t' = ih$ である．

5-2 基本的な差分微分方程式 [MO 参照]

多くの待ち行列理論は差分微分方程式系が適用されるモデルに合致する．この方程式の原型は物理学において多くの理論の基礎となっている保存則に類似している．われわれはまずそのような方程式を導出し，次に少なくとも平衡状態の場合についてそれを解こう．解の特別な場合として，前に述べた Poisson 到着，複数の並列窓口システム，タイムシェアリング・システムの非常に簡単なモデルなどがあげられる．

まずいくつかの記法(この章の初めの**図 5-1.1** も参照のこと)を掲げる．

n　システム内の客の総数

$P_j(t)$　時刻 t でシステムが状態 j である確率．状態 j とは現在処理を受けている1人を含め，待ち行列に j 人の客がいることを示す．任意の時刻 t でシステムはどれかの状態にあるので次式が得られる．

$$\sum_{j=0}^{n} P_j(t) = 1$$

(この和はある固定された時刻でのすべての可能な状態の和であり，時間に関する和で**ない**ことに注意されたい)

h　微小時間間隔

$\alpha_j h$　状態 j で1人の客が時刻 t から $t+h$ の間に待ち行列に加わる確率．α_j はシステムが状態 j にあるときの新たな要求がなされる**率**である．

$\omega_j h$　状態 j で1人の客が時刻 t から $t+h$ の間に待ち行列を抜け出す確率．ω_j はシステムが状態 j にあるときのある客が処理を終える**率**である．

一般に α_j と ω_j は j の関数であるが，多くの興味ある問題においてはそれらは j について独立であるとされる．もし時間間隔 h が充分小さいとすれば h 秒の間に待ち行列に(を)，1人の客が加わる(抜け出す)かあるいは誰も加わら

(抜け出さ)ないかのどちらかである，すなわち，2人以上の客がそうする確率は0と仮定できるであろう．

上の仮定の下に，どのようにしてシステムは時刻 $t+h$ で状態 j になりうるかを考えよう．0人または1人の客が到着したり抜け出した結果，状態 j になりうるのは，状態 $j-1$, j, $j+1$ からだけなので，時刻 t でのそれらの状態からの遷移のみを考えればよい．そのような遷移のすべての可能性は**図 5-2.1** に要約されている．遷移確率を得るためにこれを用いると，確率 $P_j(t+h)$ は次式のような和となる．

$$\begin{aligned} P_j(t+h) = \alpha_{j-1}hP_{j-1}(t) + (1-(\alpha_j+\omega_j)h)P_j(t) \\ + \omega_{j+1}hP_{j+1}(t) + h^2f' \end{aligned} \tag{1}$$

ここで h^2 を含むすべての項は h^2f' としてまとめられている．両辺から $P_j(t)$ を引き h で割って，$h\to 0$ の極限をとれば次式が得られる．

$$P_j'(t) = \alpha_{j-1}P_{j-1}(t) - (\alpha_j+\omega_j)P_j(t) + \omega_{j+1}P_{j+1}(t) \tag{2}$$

ここで $P_j'(t)$ は $P_j(t)$ の時間微分である．特別の場合として $j=0$ のときを考える．$\alpha_{-1}=0$, 待ち行列が空のとき客が待ち行列を抜け出す確率は0なので $\omega_0=0$, ゆえに

$$P_0'(t) = -\alpha_0P_0(t) + \omega_1P_1(t) \tag{3}$$

状態 前	状態 後	可能な遷移	確率表現	h の巾数による表現
$j-1$	j	1到着 と 0立去り	$\alpha_{j-1}h(1-\omega_{j-1}h)$	$\alpha_{j-1}h - h^2f_{j-1}$
j	j	0到着 と 0立去り または 1到着 と 1立去り	$(1-\alpha_jh)(1-\omega_jh)$ $+$ $\alpha_jh\omega_jh$	$1-(\alpha_j+\omega_j)h^2+2h^2f_j$
$j-1$	j	0到着 と 1立去り	$(1-\alpha_{j+1}h)\omega_{j+1}h$	$\omega_{j+1}h - hf_{j+1}$

注：f は積 $\alpha\omega$ を示す．

図 5-2.1 状態 j への可能なすべての遷移

となる．式(2)と(3)は多くの待ち行列理論の基本的な差分微分方程式(tについて微分，jについて差分)である．0からnの各jの値に対し1つの方程式があるので，方程式(2)，(3)は実際に連立方程式の系の原型である．この章の残りでは，これらの方程式の一般的な平衡状態の解と興味ある特別な場合に対するそれらの応用について考える．

5-3　平衡状態における待ち行列の振舞［MO, CO 参照］

待ち行列の振舞の1つの型として“平衡状態”がある．これは，システムの立ち上りの過渡状態がなくなった後で，時間に関して状態確率が変化しなくなった時間領域として定義される．したがって平衡状態ではすべてのj, tに関して$P_j'(t)=0$である．前節の式(3)にこの条件を入れて

$$P_1=\frac{\alpha_0}{\omega_1}P_0 \tag{1}$$

を得る．またP_{j+1}についての前節の式(2)は

$$P_{j+1}=\left(\frac{\alpha_j+\omega_j}{\omega_{j+1}}\right)P_j-\left(\frac{\alpha_{j-1}}{\omega_{j+1}}\right)P_{j-1} \tag{2}$$

となる．

jに次々といくつか値を代入してみると，式(2)の解として次式が得られることがわかる．

$$P_j=\left(\frac{\alpha_0\alpha_1\cdots\alpha_{j-1}}{\omega_1\omega_2\cdots\omega_j}\right)P_0=\frac{\alpha_0}{\omega_j}\left(\prod_{i=1}^{j-1}\frac{\alpha_i}{\omega_i}\right)P_0 \tag{3}$$

P_0はすべてのjについてP_jを加えあわせると得られる．もちろんその和は1となる．P_0について解けば

$$P_0=\frac{1}{1+\dfrac{\alpha_0}{\omega_1}+\displaystyle\sum_{j=2}^{n}\frac{\alpha_0}{\omega_1}\left(\prod_{i=1}^{j-1}\frac{\alpha_i}{\omega_i}\right)} \tag{4}$$

となる．式(3)，(4)は基本となる平衡状態待ち行列式であり，本章の後の方ではこの式のいくつかの特別な場合について考える．

待ち行列長の平均(あるいは期待値)は式(3)を用いて得られ

$$\bar{j}=\sum_{j=0}^{n}jP_j=\left(\frac{\alpha_0}{\omega_1}+2\frac{\alpha_0\alpha_1}{\omega_1\omega_2}+3\frac{\alpha_0\alpha_1\alpha_2}{\omega_1\omega_2\omega_3}+\cdots\right)P_0 \tag{5}$$

である． 式(3)から(5)の特別な場合として，すべての α_j が α に等しく，すべての ω_j が ω に等しいときを考える．到着率と処理率の比を定義しよう．

$$r=\frac{\alpha}{\omega}=\frac{1\text{人の客の平均処理時間}}{\text{次の客の到着までの平均時間}} \tag{6}$$

この比は**トラフィック密度**(traffic intensity) と呼ばれ， 待ち行列理論の先駆者 A. K. Erlang の名をとってその単位を**アーラン**と呼ぶ．

待ち行列の長さが j である確率は式(3)から

$$P_j=r^jP_0 \tag{7}$$

となる．また

$$\sum_{j=0}^{n}P_j=1 \tag{8}$$

である．式(7)の P_j を代入し，幾何級数の公式を用いると

$$P_0=\frac{1-r}{1-r^{n+1}} \tag{9}$$

となる．式(9)を式(7)に代入して次式を得る．

$$P_j=r^j\frac{1-r}{1-r^{n+1}}$$

待ち行列の平均長(処理中の客も含める)は式(10)を式(5)に代入して得られ，代数的に簡略化すれば

$$\bar{j}=\frac{r}{1-r^{n+1}}\,\frac{1+r^n(nr-n-1)}{1-r} \tag{11}$$

となる． この式に対し， 文献2の18ページにあるように次のような興味ある3つの場合が考えられる．

$$r\ll 1:\quad \bar{j}=r+r^2 \tag{12 a}$$

$$r\to 1:\quad \bar{j}=\frac{n}{2}+\frac{n(n+2)(r-1)}{12} \tag{12 b}$$

$$r\ll 1:\quad \bar{j}=n-\frac{1}{r} \tag{12 c}$$

待ち行列の長さの分散(平均からの平均2乗差)は

$$\nu = \sum_{j=0}^{n} (j - \bar{j})^2 P_j = \sum_{j=0}^{n} j^2 P_j - \bar{j}^2 \tag{13}$$

となる．上と同じ3つの場合の r に対してこの式は次のように変形される．

$$r \ll 1: \quad \nu = r + 2r^2 \tag{14 a}$$

$$r \to 1: \quad \nu = \frac{1}{12} n(n+2) \tag{14 b}$$

$$r \gg 1: \quad \nu = \frac{1}{r} + \frac{2}{r^2} \tag{14 c}$$

5-4　平衡状態解：長さに制限のない待ち行列

客の数 n，それゆえ，待ち行列の長さとして考えられる最大値，が限りなく大きくなりうるとすれば，その結果かなり簡略化が行なわれる．前節の式(10)から始めることにするが，ここではそれを式(1)として再掲する．

$$P_j = \frac{r^j(1-r)}{1-r^{n+1}} \tag{1}$$

ここで $r = \dfrac{\alpha}{\omega}$ である．

$r < 1$ で $n \to \infty$ としてもよいと仮定する．このとき式(1)は次のようになる．

$$P_j = r^j(1-r) \tag{2}$$

P_0 は待ち行列に客がいない．すなわち，窓口の機能が遊んでいる確率であるから，$1 - P_0$ は窓口が使用中である(利用されている)確率である．

$$\rho = 1 - P_0 = 1 - (1-r) = r = \frac{\alpha}{\omega} \tag{3}$$

このように，窓口の利用率 ρ はトラフィック密度 r に等しいことがわかる．

待ち行列長の期待(平均)値は式(2)を用いて

$$\bar{j} = E(j) = \sum_{j=1}^{\infty} jr^j(1-r) = (1-r)\sum_{j=1}^{\infty} jr^j \tag{4}$$

で与えられる．一般的な公式によると

$$\sum_{j=1}^{n} jr^j = \frac{r - r^{n+1}}{1-r} \cdot \frac{1}{1-r} - \frac{nr^{n+1}}{1-r} \tag{5}$$

であり，$n \to \infty$ のとき r^n の項はすべて0に収束するので

$$\sum_{j=1}^{\infty} jr^j = \frac{r}{(1-r)^2} \tag{6}$$

が得られる．式(6)を式(4)に代入し，式(3)を用いて，平均待ち行列長は

$$\bar{j} = \frac{r}{1-r} = \frac{\rho}{1-\rho} \tag{7}$$

で与えられる．

平均行列時間 $\bar{t}_q$ を求めるために，平衡状態で満足されなければならない次の関係を利用する．

$$\alpha \bar{t}_q = \bar{j} \tag{8}$$

j に対し式(7)を用い，$r = \alpha/\omega$ の関係によって次式が得られる．

$$\bar{t}_q = \frac{1}{\omega(1-r)} = \frac{\bar{t}_s}{1-r} \tag{9}$$

ここで $\bar{t}_s = 1/\omega =$ 平均処理時間である．

平均待ち時間は待ち行列で待つ平均時間で，処理中の時間を含めないものと定義され

$$\bar{t}_\omega = \bar{t}_q - \bar{t}_s \tag{10}$$

となり，$\bar{t}_q$ に関する式(9)を用いると

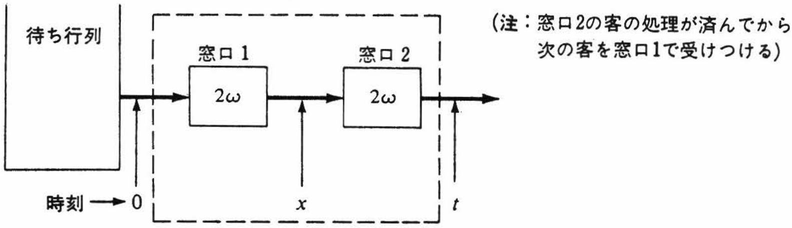
$$\bar{t}_\omega = \frac{\bar{t}_s}{1-r} - \bar{t}_s = \frac{r\bar{t}_s}{1-r} \tag{11}$$

となる．式(7)(9)(11)から利用率が1に近づくと平均待ち行列長，平均待ち時間は急速に大きくなることがわかる．言い換えれば，処理量をどんどん増やして，高利用率という経済的利点にあまりにも重点を置くと，1に近い利用率の代償として平均待ち時間が非常に長くなる．

5-5 Erlang 分布と超指数分布：直列の窓口

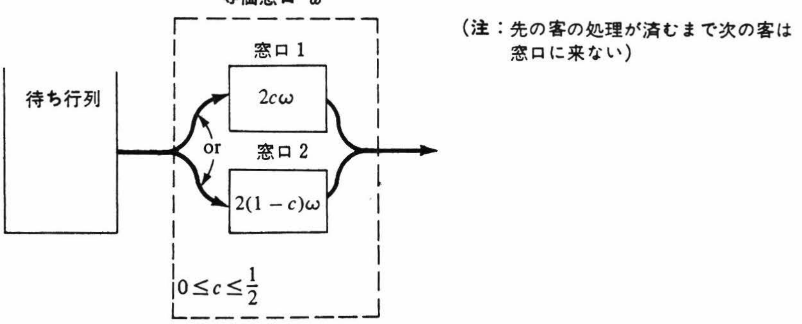
待ち行列問題の解析的な解は到着と処理時間が指数分布の場合についてのみしか一般には得られないが，実際には，特に処理時間については，この型の分布は一般的でないと前に述べた．それゆえ，まず他の型の分布関数を記述することに興味がもたれる．次に，もしそれが可能なら，それらを指数を用いて記述できるかに興味がある．これを行なう1つの方式を以下説明しよう．これは待ち行列理論の先駆者 A. K. Erlang によるものである．

1つの待ち行列は k 個の窓口で次の規則にしたがって順に処理を受けるものとする．待ち行列から選ばれたジョブはまず第1の窓口に送られ，そこで処理を受けたあと，第2の窓口へ送られる．このようにして次々と窓口を進み，最後に第 k 窓口から抜け出す．この後初めて，待ち行列中の次のジョブが第1窓口に送られる．前のジョブが k 個の窓口全部を通り過ぎて初めて次のものが開始されるので，k 個の窓口の直列に並んだもの全体はある等価は1つの窓口に対応すると考えられる．この等価な窓口の処理時間分布を導き出そう．図



等価窓口の性質：$\bar{t}_s = \frac{1}{\omega}$　$\sigma^2 = \frac{1}{k\omega^2}$

(*a*) アーラン分布に対する直列窓口($k = 2$)



等価窓口の性質：$\bar{t}_s = \frac{1}{\omega}$　$\sigma^2 = \frac{1}{\omega^2}\left[1 + \frac{(1-2c)^2}{2c(1-c)}\right]$

(*b*) 超指数分布に対する択一的窓口

図 5-5.1　アーラン分布と超指数分布の複数窓口

5-5.1 は最も簡単な $k = 2$ の場合であり，どのように時刻にラベルを付けるかを示している．時刻0は窓口1の開始時刻であり，x は窓口1の処理の終る時刻(また窓口2の開始時刻)であり，そして t は窓口2の完了時刻である．時刻0と t はまた等価な窓口の開始および完了時刻である．2つの窓口は同じ指数分布の処理時間で，それぞれ平均処理率を 2ω(ω でなく 2ω である理由は後でわかるであろう)と仮定する．上で書かれたラベルを使えば，第1の窓口の処理が終了する確率は

$$2\omega e^{-2\omega x}dx \tag{1}$$

であり，また第2の処理の完了の確率は次式となる．

$$2\omega e^{-2\omega(t-x)}dt \tag{2}$$

ジョブが2つの窓口の両方(1つの等価な窓口)を終える確率，すなわち時刻 t から $t + dt$ の間に抜け出す確率は式(1)，(2)の積の x に関する積分となる．

$$\begin{aligned} s(t)dt &= dt\int_0^t (2\omega)^2 e^{-2\omega x}e^{-2\omega(t-x)}dx \\ &= (2\omega)^2 e^{-2\omega t}dt\int_0^t dx \end{aligned} \tag{3 a}$$

または

$$s(t)dt = (2\omega)^2 te^{-2\omega t}dt \tag{3 b}$$

となる．式(3 b)はあるジョブが時刻 t から $t + dt$ の間に2つの窓口の処理を完了する確率であるから，ジョブが0から t の間に完了する確率は

$$\int_0^t (2\omega)^2 te^{-2\omega t}dt \tag{4}$$

である．そこで0から t までの間に1つのジョブも完了しない確率は次式で与えられる．

$$S_0(t) = 1 - \int_0^t 4\omega^2 te^{-2\omega t}dt = \int_0^\infty 4\omega^2 te^{-2\omega t}dt - \int_0^t 4\omega^2 te^{-2\omega t}dt \tag{5}$$

あるいは

$$\begin{aligned} S_0(t) &= \int_t^\infty 4\omega^2 te^{-2\omega t}dt = \int_{2\omega t}^\infty ye^{-y}dy \\ &= (1 + 2\omega t)e^{-2\omega t} \end{aligned} \tag{6}$$

この分布関数は t の積となるので指数関数ではない．平均処理時間は次のよ

うに定義される.

$$\bar{t}_s = \int_0^\infty ts(t)dt = \frac{1}{\omega} \tag{7}$$

各窓口の平均処理率を 2ω とした理由がわかるであろう. これにより等価窓口の処理率は ω となる.

処理率がおのおの $k\omega$ の窓口が k 個直列に並ぶもっと一般的な場合にも上と(詳細にはもっと複雑であるが)同様の方法で次式の Erlang 分布関数が等価窓口に適用されることがわかる(文献2).

$$t_s(t)dt = (k\omega t)^{k-1}\left(\frac{e^{-k\omega t}}{(k-1)!}\right)k\omega tdt \tag{8 a}$$

これから

$$S_0(t) = e^{-k\omega t}\sum_{i=0}^{k-1}\frac{(k\omega t)^i}{i!} \tag{8 b}$$

ここで

$$\begin{aligned} \bar{t}_s &= \frac{1}{\omega} \qquad \text{(平均)} \\ \sigma &= \frac{1}{\omega\sqrt{k}} \qquad \text{(標準偏差)} \end{aligned} \tag{9}$$

である. $k=1$ に対しては Erlang 関数は単に指数関数となり, したがってその分散は平均に等しい. $k\to\infty$ のとき, 分布は $t \leq \dfrac{1}{\omega}$ では定数1に, $t > \dfrac{1}{\omega}$ では0に近づく. したがってその分散は0となる. 中間の k の各値に対して, Erlang 分布はある中間の分散を待ち t に関して滑かな曲線となる. (測定によって概略的に得られる) 実際の分布で平均より分散が小さいものに対しては, k を適当に選ぶことによりしばしば Erlang 関数にほぼ合致させうる.

Erlang 関数を導出するために用いられた図5-5.1の窓口の直列的な並びはそんなに厳密に考えられるべきものでは**ない**という点を強調しなければならない. モデル化される実際のシステムは, Erlang 分布を用いるために複数の窓口が順に並んだものから成る必要はない. 実際に要求されることは Erlang 分布が実際の分布にかなり良く合致するようにさせることである.

指数分布の組合せから得られるもう1つの型の分布関数は**超指数**分布である. これは Erlang 分布で表わせない場合, すなわち, 処理(あるいは到着)分布の分散がその平均より**大きい**場合を表現できる. この分布は1つの待ち行列が2つの**択一的**な指数窓口により処理を受ける場合を想定することにより導き

出される．その 2 つの窓口の処理率はそれぞれ $2c\omega$ と $2(1-c)\omega$ であり c は

$$0 < c \leqslant 1/2$$

の範囲のある値(後で選ばれる)である．上で述べたように 2 つの窓口には 1 つの待ち行列から次のように客が来る． 2 つの窓口に仕事がないとき，待ち行列にある次のジョブは片方の窓口， あるいは他の窓口にそれぞれ確率 c， $1-c$ で割付けられる．この後，そのジョブが完了するまで他のジョブ割付けはなされない． 2 つの窓口は各ジョブに択一的に割付けられる可能性のあるもので，一時にただ 1 つの窓口しかは仕事をしない．したがって，この構成は 1 つの等価窓口として考えられ，この方式の処理時間密度分布を記述することはかなり容易である．択一的な各窓口の確率密度を求め，それにそれが選ばれる確率をかけ，積の和をとると次式となる．

$$s(t) = c(2c\omega e^{-2c\omega t}) + (1-c)(2(1-c)\omega e^{-2(1-c)\omega t}) \tag{10}$$

この分布の平均と分散は次のようになる．

$$\bar{t}_s = \frac{1}{\omega}$$

$$\sigma^2 = \frac{1}{\omega^2}\left(1 + \frac{(1-2c)^2}{2c(1-c)}\right) \tag{11}$$

時刻 0 から t までの間につ 1 の処理も 完了しない確率は $s(t)$ の積分で得られる．

$$S_0(t) = ce^{-2c\omega t} + (1-c)e^{-2(1-c)\omega t} \tag{12}$$

5-6 Khintchine-Pollaczek 方程式 [CO, MA 参照]

これらの平衡状態の待ち行列問題の解は以下のかなり一般的な条件の下に成立する．

1. Poisson 到着時間(= 指数的到着時間間隔)
2. **任意**の処理時間分布
3. 処理時間に関知**しない**任意の待ち行列規則(スケジューラ)

待ち行列の平均長の基本的な式は次のようになる．

$$\bar{j} = r + \frac{r^2}{2(1-r)}\left(1 + \left(\frac{\sigma_s}{\bar{t}_s}\right)^2\right) \tag{1}$$

ここで，

$$r=\frac{\alpha}{\omega}$$

$$\frac{1}{\omega}=\bar{t}_s=\text{平均処理時間}$$

$$\sigma_s=\text{処理時間の標準偏差}$$

いくつかの関係式が式(1)から非常に容易に得られる．まず少しばかりの記法と公式をつけ加えよう．

$\bar{t}_q$ = 待ち行列中の平均時間(処理中の時間も含める)

$\bar{t}_w$ = 平均待ち時間(処理時間を含まない待ち行列中の時間)

$\bar{j}$ = 待ち行列の平均長(処理中のものを含む)

$\bar{k}$ = 待ち行列の平均長(処理中のものを含まない)

そして，

$$\bar{t}_q=\bar{t}_w+\bar{t}_s \tag{2}$$

$$\bar{k}=\alpha\bar{t}_w \tag{3}$$

$$\bar{j}=\alpha\bar{t}_q=\alpha(\bar{t}_w+\bar{t}_s)=\bar{k}+\alpha\bar{t}_s=\bar{k}+r \tag{4}$$

$$\bar{k}=j-r \tag{5}$$

これらを用いて，図5-6.1の一般公式が導き出される．

	一般式	Poisson 処理時間 $\sigma_s=\bar{t}_s$	一定処理時間 $\sigma_s=0$
平均待ち行列長	$j=r+\dfrac{r^2}{2(1-r)}\left(1+\left(\dfrac{\sigma_s}{\bar{t}_s}\right)^2\right)$	$\bar{j}=\dfrac{r}{1-r}$	$\bar{j}=\left(\dfrac{r}{1-r}\right)\left(1-\dfrac{r}{2}\right)$
平均待ち行列時間	$\bar{t}_q=\bar{t}_s\left(\dfrac{1+r}{2(1-r)}\left(1+\left(\dfrac{\sigma_s}{\bar{t}_s}\right)^2\right)\right.$	$\bar{t}_q=\dfrac{\bar{t}_s}{1-r}$	$\bar{t}_q=\dfrac{\bar{t}_s}{1-r}\left(1-\dfrac{r}{2}\right)$
平均待ち時間	$\bar{t}_w=\dfrac{\bar{t}_s}{2(1-r)}\left(1+\left(\dfrac{\sigma_s}{\bar{t}_s}\right)^2\right)$	$\bar{t}_w=\bar{t}_s\left(\dfrac{r}{1-r}\right)$	$\bar{t}_w=\bar{t}_s\left(\dfrac{r}{2(1-r)}\right)$

注：　同じ作業負荷パラメータに対し，Poisson 処理時間はより悪い性能を与える．

図 5-6.1　Khintchine-Pollaczek 方程式を用いての一般式と特殊な場合の比較

5-7　並列窓口システム［CO 参照］

これまでシステムは1つの窓口を持っていた．さてその結果を m 個の窓口の場合に一般化しよう．平衡状態での状態 j の確率を導出するため，第5-3節の式(3)，(4)の一般解を考えよう．もし j(待ち行列の長さ)と m(窓口の数)の関係が $j\leqslant m$ ならば，一度に待ち行列のすべてが処理されるのに充分な窓口

があり，実効処理率は $j\omega$($\omega=$ 1つの窓口の処理率)である．しかし待ち行列の長さが m より大きいと，窓口は飽和し，実効処理率は $m\omega$ である．すべての場合 α_j は j に独立で $\alpha_j=\alpha$ と仮定する．

$$\alpha_j=\alpha \tag{1 a}$$

$$\omega_j=j\omega \qquad 0\leqslant j\leqslant m \tag{1 b}$$

$$\omega_j=m\omega \qquad m\leqslant j\leqslant n \tag{1 c}$$

第5-3節の一般式にこれらの式を慎重に代入し，少し面倒な代数的簡略化を行なって次式を得る．

$$P_j=\frac{(mr)^j}{j!}P_0 \qquad 0\leqslant j\leqslant m \tag{2}$$

$$P_j=\frac{m^m r^j}{m!}P_0 \qquad m\leqslant j\leqslant n \tag{3}$$

ここで　$r=\dfrac{\alpha}{m\omega}$ である．

これらの式の利用の1つは，**1つの**高速な窓口を持つシステムと複数の低速な窓口を持つシステムとの興味深い比較を行なうことである．具体的には処理率 $m\omega$ の1つの窓口を持つシステムを仮定し，このシステムの P_j と，それぞれの処理率が ω の m 個の窓口を持つシステムの P_j とを比較する．最初の単一窓口の待ち行列の確率は r を $\alpha/m\omega$ として第5-3節の式 (10) から直接得られる．m 個窓口システムでの対応する P_j は式(2)，(3)によって与えられる．後者は前者に $m^j/j!$ または $m^m/m!$ をかけたものと考えられ，これらの係数は共に1より大である．複数窓口の場合の P_0 は高速な単一窓口の場合の P_0 より常に大きいことがわかるであろう．第4章の簡単な推論によって次の結果が得られる．

単一窓口システムの処理率を m 倍改善したほうが処理率はそのままで m 個の並列窓口とするよりよい結果が得られる．

この結果はこのモデルで，この性能尺度を用いたときに適用されるもので，すべての可能な単一窓口と並列窓口との比較における一般的な結果では**ない**．

5-8 タイムシェアリング・システム・モデル [SC 参照]

第5-3節の平衡状態の一般解に現れる変数を適切に解釈すると，あるタイムシェアリング・システムとの有益な類似点を持つ簡単なモデルを解くことがで

きる．

契約した n 人のユーザを仮定する．任意の時刻に各ユーザは2つの可能な状態，**考慮中**と**待ち**，のいずれか一方のみに居ることができる．考慮状態にあるとき，ユーザは処理を(実際のシステムでは1行タイプすることで)要求することができる．ユーザは要求し終った時点で，待ち状態に入る．すなわち彼はシステムによる処理を待っている．待ち状態の時間には計算機の順番が彼に回ってくるのを“待っている”時間だけでなく，彼の要求を処理するために必要な時間も含める．待ち状態中はユーザは他の要求を出すことを禁じられている．(一般には彼のキーボードがロックされているから)．システムが要求に対する処理を終えたとき，ユーザは待ち状態から離れ，再び考慮状態に入る．次の記法が用いられる．

$\alpha = 1/T =$ ある与えられた**1人の**ユーザが処理を要求する平均の率，そのような要求は考慮状態にあるときのみ可能であるから，T は平均考慮時間と考えられる．

$\alpha h = h/T =$ ある与えられた1人のユーザが微小時間間隔 h の間に処理を要求する確率

$\alpha_j =$ システムが状態 j(待ち行列中に j 個の要求がある状態)にある時，あるユーザ(すなわち，**任意の**ユーザで，指定されたユーザでない)が処理を要求する平均の率

$1/C =$ システムが要求の処理を終える平均の率

C は平均処理時間と考えられる．ユーザが n 人のシステムが状態 j にあるとしよう．これは待ち行列中に j あり，待ち行列外に $n-j$ あることを意味する．すなわち，後者の $n-j$ は考慮状態にある．われわれの式を用いるため，$\alpha_j h$ の定義に注意しよう．その定義によると，待ち行列外の $n-j$ のユーザだけが要求を出せる候補であるから，$n-j$ 人のユーザのある1人が時間間隔 h の間に要求を出す確率は，等しい可能性のある1人1人の確率の $n-j$ 回の和である．

$$\alpha_j h = \frac{(n-j)h}{T} \qquad \text{または} \qquad \alpha_j = \frac{n-j}{T} = (n-j)\alpha \tag{1}$$

時間間隔 h の間にユーザが待ち行列から離れる．すなわち，処理が完了する確率は

$$\omega_j h = \frac{h}{C} \qquad \text{または} \qquad \omega = \frac{1}{C} \tag{2}$$

である．このモデルはタイム・スライシングを考慮していないことに注目すべきである．

上の仮定のもとに，α_j と ω に関する式(1)，(2)を5-3節の式(3)に代入すると

$$P_j = P_0\left(\frac{n\alpha}{\omega}\right)\left(\frac{(n-1)\alpha}{\omega}\right)\left(\frac{(n-2)\alpha}{\omega}\right)\cdots\left(\frac{(n-j+1)\alpha}{\omega}\right) \quad (3)$$

あるいは

$$P_j = P_0\frac{n!\,r^j}{(n-j)!} \quad (3\,\mathrm{a})$$

となる．ここで

$$r = \frac{\alpha}{\omega} = \frac{C}{T} \quad (4)$$

である．P_j をすべて加えると1であるから，式(3a)をすべての j について加えると次のようになる．

$$P_0 = \frac{1}{\displaystyle\sum_{j=0}^{n}\frac{n!\,r^j}{(n-j)!}} \quad (5)$$

式(3a)に代入して

$$P_j = \frac{n!\,r^j}{(n-j)!}\cdot\frac{1}{\displaystyle\sum_{j=0}^{n}\frac{n!\,r^j}{(n-j)!}} \quad (6)$$

となる．待ち行列の平均長は定義より

$$\bar{j} = \sum_{j=0}^{n} jP_j \quad (7)$$

である．

平衡状態での待ち行列システムでは，待ち行列の平均長と平均待ち時間の関係の一般式は次で表わされる．

$$\bar{\alpha}\bar{t}_q = \bar{j} \quad (8)$$

ここで

$\bar{\alpha}$ = 待ち行列への平均到着率

$\bar{t}_q =$ 待ち行列中の平均時間

$\bar{j} =$ 待ち行列の平均長

である．式(8)の直観的な意味は，新しい客が平均 $\bar{\alpha}$ の率で到着し，待ち行列中の平均時間 $\bar{t}_q$(処理時間をも含む)を持つなら，待ち行列の長さは平均として $\bar{\alpha}$ と $\bar{t}_q$ の積になるということである．$\bar{\alpha}$ の定義は α や α_j と混同されるべきでなく，

$$\bar{\alpha} = \sum_{j=0}^{n} \alpha_j P_j = \sum_{j=0}^{n} \frac{n-j}{T} P_j \tag{9}$$

である．2番目の総和の式には α_j に関する式(1)を用いた．P_j の和が1であることを用いて，さらに式(9)を簡略化すれば次式を得る．

$$\bar{\alpha} = \frac{n}{T} - \frac{1}{T}\sum_{j=0}^{n} jP_j = \frac{n}{T} - \frac{\bar{j}}{T} \tag{10}$$

変形には j に関する式(7)を用いた．式(10)を式(8)に代入し $\bar{t}_q$ について解けば次式となる．

$$\bar{t}_q = \frac{\bar{j}\,T}{n-\bar{j}} \tag{11}$$

添字の処理と和の極限など少し面倒な代数処理により（詳しくは章の終りの補遺5-Aに示されている)，式(11)は次のように簡略化される．

$$\frac{\bar{t}_q}{C} = \frac{n}{1-P_0} - \frac{1}{r} \tag{12}$$

P_0 に関する式(5)を用いて

$$\frac{\bar{t}_q}{C} = \frac{n}{1 - \dfrac{1}{\displaystyle\sum_{j=0}^{n} \frac{n!\, r^j}{(n-j)!}}} - \frac{1}{r} \tag{13}$$

となる．$\bar{t}_q/C$ に対する n と r の表を直接計算することは可能である．しかしこの表のもっと簡単な再帰的形式（章の終りの補遺5-Bにより導出される）によりこの計算はずっと速くなる．この再帰的な式は次のようである．

$$Y(n+1) = (n+1) - \frac{n}{1+rY(n)} \tag{14}$$

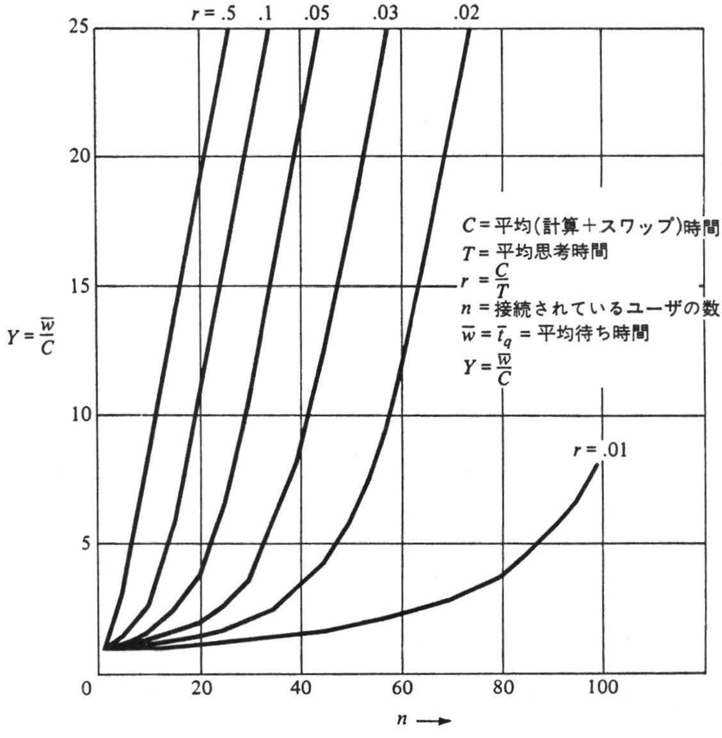
$$Y(1) = 1$$

$$Y(1) = 1 \tag{15}$$

ここで　$Y = \bar{t}_q/C$ である

平均待ち，平均処理，平均考慮時間に関することの公式を接続されているユーザの数(n)に対してプロットしたものを図 5-8.1 に示す.

最後に，上のモデルは(この章でのモデルすべてがそうであるように)第 3-17 節で議論された Markov 連鎖法によってでも解析されることを注意しておかなければならない.



Yに対する再帰式

$Y(1) = 1$

$Y(n+1) = (n+1) - \dfrac{n}{1 + rY(n)}$

For $n >> \dfrac{1}{r}$, $Y(n) \approx n - \dfrac{1}{r}$

図 5-8.1 タイムシェアリング待ち行列モデルの正規化された平均待ち時間

補　遺　V-I
第5-8節式(11)から式(12)の導出

（注：　$\bar{t}_q$ の省略形として $\bar{w}$ を用いる）

$$\bar{w} = \frac{Tj}{n-j} \tag{1}$$

$$\bar{j}(n) = \sum_{j=0}^{n} jP_j \sum_{j=0}^{n} P_j(n) = 1 \tag{2}$$

$$n - \bar{j}(n) = \left(n\sum_{j=0}^{n} P_j(n)\right) - \sum_{j=0}^{n} jP_j(n) \tag{3}$$

$$\sum_{j=0}^{n} (n-j)P_j = P_0(n)\sum_{j=0}^{n} (n-j)\frac{n!\,r^j}{(n-j)!} \tag{3a}$$

和の最後の項＝0 であるから

$$n - \bar{j}(n) = P_0(n)\sum_{j=0}^{n-1} (n-j)\frac{n!\,r^j}{(n-j)!} = P_0(n)\sum_{j=0}^{n-1} \frac{n!\,r^j}{(n-j-1)!}$$

$$= nP_0(n)\sum_{j=0}^{n-1} \frac{(n-1)!\,r^j}{(n-j-1)!} \tag{4}$$

和は $1/P_0(n-1)$ であるので

$$n - \bar{j}(n) = \frac{nP_0(n)}{P_0(n-1)} \qquad \text{または } (n)\bar{j} = n\left(1 - \frac{P_0(n)}{P_0(n-1)}\right) \tag{5}$$

この式を(1)に代入し，簡略化すれば

$$\bar{w} = T\left(\frac{P_0(n-1)}{P_0(n)} - 1\right) \tag{6}$$

となる．さて比 $P_0(n-1)/P_0(n)$ を簡略化しよう

$$P_0(n) = \frac{1}{\displaystyle\sum_{j=0}^{n} \frac{n!\,r^j}{(n-j)!}} \tag{7}$$

$$P_0(n-1) = \frac{1}{\displaystyle\sum_{j=0}^{n-1} \frac{(n-1)!\,r^j}{(n-j-1)!}} \tag{8}$$

ここで $k=j+1$ とする．$j=k-1$ であり

$$P_0(n-1)=\frac{1}{\sum_{k=1}^{n}\frac{(n-1)!\,r^{k-1}}{(n-k)!}}=\frac{1}{\frac{1}{rn}\sum_{k=1}^{n}\frac{n!\,r^k}{(n-k)!}}$$

$$=\frac{rn}{\frac{1}{P_0(n)}-1} \tag{9}$$

これから

$$P_0(n-1)=\frac{rnP_0(n)}{1-P_0(n)} \tag{10}$$

となる．式(6)に代入すれば

$$\bar{w}=T\left(\frac{rn}{1-P_0(n)}-1\right)=\frac{Cn}{1-P_0(n)}-T=\frac{Cn}{1-P_0(n)}-\frac{C}{r} \tag{11}$$

となる．式(11)と全く等価な $\bar{w}(n)$ のもう1つの式は

$$\bar{w}(n)=C(1+\bar{j}(n-1)) \tag{12}$$

である．

補　遺　V-II
平均待ち時間(5-8 節)のための再帰的形成

第5-8節の式(12)から始める．

$$Y(n)=\frac{n}{1-P_0(n)}-\frac{1}{r} \tag{1}$$

$P_0(n)$について解く．

$$P_0(n)=1-\frac{nr}{1+rY(n)} \tag{2}$$

式(1)より

$$Y(n+1)=\frac{n+1}{1-P_0(n+1)}-\frac{1}{r} \tag{3}$$

5-8 節の式(5)の P_0 を用い，$j-1$ の代りに q を代入すれば

$$\frac{1}{P_0(n+1)}=\sum_{j=0}^{n+1}\frac{(n+1)!\,r^j}{(n-j+1)!}=r(n+1)\sum_{q=-1}^{n}\frac{n!\,r^q}{(n-q)!} \tag{4}$$

右辺の和を $q=-1$ の項と残りの項の和に書き直す

$$\frac{1}{P_0(n+1)}=r(n+1)\left(\frac{1}{r(n+1)}+\sum_{q=0}^{n}\frac{n!\,r^q}{(n-q)!}\right) \tag{5}$$

和の項は $P_0(n)$ であるから式(5)は次のようになる.

$$\frac{1}{P_0(n+1)}=1+\frac{r(n+1)}{P_0(n)} \tag{6}$$

両辺の逆数を取り，項を並べかえると

$$P_0(n+1)=\frac{P_0(n)}{P_0(n)+r(n+1)} \tag{7}$$

(7)式を(3)式に代入し簡単化すれば

$$Y(n+1)=(n+1)+\frac{P_0(n)}{r}-\frac{1}{r} \tag{8}$$

となる．$P_0(n)$の代りに式(2)を代入し，簡単化すれば

$$Y(n+1)=(n+1)-\frac{n}{1+rY(n)} \tag{9}$$

ここで

$$Y(1)=1 \tag{10}$$

となる.

第5-8 節の式(13)の一般形に比較しこの再帰形は非常に簡単である.

演　習　問　題

1. 平均到着時間間隔が10秒のまったくランダムな (Poisson) 到着を仮定したとき，2, 5, 10, 20，そして100秒の時間の間に少なくとも1つの到着がある確率はいくらか？　また，それらの時間内に2つ以上の到着がある確率はいくらか？

2. 次の Poisson 到着時間間隔，処理時間を仮定する．

平均到着時間間隔 = 10 秒
平均処理時間 = 5 秒
最大待ち行列時間 = 4 秒

このとき，以下の値を求めよ．

(a) トラフィック密度
(b) 平均待ち行列時間
(c) 待ち行列時間偏差
(d) 利用率

3. 問題2で待ち行列時間の最大値に制限を与えない場合の答を求めよ．

4. ある計算機設備では平均ジョブ待ち時間が10分で，システム利用率は40% である．指数(Poisson)関数を仮定すれば，平均処理時間はいくらか？

5. ある計算機センタではジョブの平均実行時間が3分であることがわかっている．また実行時間の分布関数は指数分布であるとされている．指数分布の妥当性を調べるために直観的洞察とともに用いられる簡単な計算式を示せ．

6. 平均待ち時間と平均処理時間の比は“平均の引き伸し率”と考えられる．Poisson 関数と最大待ち行列長が無限大での定常状態条件を仮定し，トラフィック密度値 $0 \leqslant r < 1$ に対して平均の引き伸し率の曲線を描け．

7. タイムシェアリング・システムの応答性の適切な尺度としての平均待ち(応答)時間の値について厳密に論ぜよ．タイムシェアリング・システムの充分な議論に関しては9章を参照せよ．

8. FCFS スケジューラを用いているあるシステムでは，到着はランダムで平均が6秒である．処理時間分布は知られていないが，処理時間間隔の平均は2秒で分散係数は0.3である．

(a) 平均待ち時間を求めよ．
(b) 上記と同じ到着時間間隔と処理時間の平均を持ち，分布が指数分布であると仮定する．このときの平均待ち時間を求めよ．

9. 次の値を持つタイムシェアリング・システムを考える．

25 = 接続されているユーザ数

15秒 = 平均思考時間

1秒 = 平均計算時間

（a）　平均待ち時間を求めよ

（b）　もし平均待ち時間が平均計算時間の5倍しか許されないとすればどれだけの数のユーザが接続されるか？

参考文献

1. [CO] Conway, R. W., W. L. Maxwell, and L. W. Miller: "Theory of Scheduling," Addison-Wesley Publ. Co., Reading, Mass., 1967.
2. [CO] Cox, D. R., and W. L. Smith: "Queues," Methuen & Co., London, 1961.
3. [GO] Gordon, G.: "System Simulation," Prentice-Hall, New York, 1969.
4. [MA] Martin, J.: "Design of Real-Time Computer Systems," Prentice-Hall, New York, 1967.
5. [MO] Morse, P. M.: "Queues, Inventories, and Maintenance," John Wiley and Sons, New York, 1958.
6. [RA] Rasch, P. J.: A Queuing Theory Study of Round-Robin Scheduling of Time-Sharing Systems, *J. ACM*, vol. 17, no. 1, January 1970.
7. [SC] Scherr, A. L.: "An Analysis of Time-Shared Computer Systems," MIT Press, Cambridge, Mass., 1967.

第 6 章

単一構成要素-単一ジョブ性能

この章では，まず主記憶装置とか CPU などの構成要素の1つ1つの性能について考え，そして次に I/O と CPU 動作は重複できるが，一度に1つのジョブしか扱えないと制限されたシステムのスループット・モデルを考える．複数のジョブの間で資源の共有がない場合でさえ，システムの振舞は複雑で変化に富み，その性能の問題は簡単でない．この章で議論する論題と論争点は次の目的に役立つであろう．

1. 多くの小・中規模のシステムの場合がそうであるように，一度に1つのジョブを実際に処理するシステムを解析する．
2. ある環境での CPU 速度などの性能の性質を理解する．その環境はそれらの性質が十分正確に定義されるほど単純である．
3. マルチプログラム・システムやタイムシェアリング・システムのようなより複雑な場合と比較するために有用ないくつかの基本的な事例を与える．

6-1 主記憶装置［HE 1 参照］

主記憶装置とは一般に CPU が命令やデータを**直接に**アクセスできる記憶装置を指す．これはかなり単純な概念で，しばしば十分に実用的な定義であるが，最近のシステムにおいては必ずしも正確なものではない．（この節の終りの議論を参照のこと）

主記憶装置の代表的な容量と速度は(1972 年現在)次の範囲にある．

サイクル時間:　500 ～ 2,000 ナノ秒

容量:　8k～2M バイト = 2k～500k 語 = 64k～16M ビット

代表的な小規模主記憶装置は1つのデータ・レジスタと1つのアドレス・レジ

スタを持ち，磁気コア，磁性薄膜，または，モノリシックな回路素子を配列結線したものとして製造される．それを**モジュール**と呼び，モジュール単位には一度に1つのアクセスしか実行されない．

大部分のコアと薄膜では，メモリからの物理的な読出しはアクセスした情報を破壊する．そのため，読出しサイクルは2つの部分サイクル，すなわちアクセスと再書込みから成る．計算機の設計では，再書込み動作の無駄な時間にもCPU を有効に働かすよう大いに工夫がなされる†．磁気コア記憶装置の1つのモジュールは次のパラメータで特徴づけられる．

1. 容量(バイト，語，または，ビット)
2. アクセス幅(一度にアクセスできるバイト数/語数/ビット数)
3. アクセス時間
4. サイクル時間
5. **帯域**(bandwidth) = アクセス幅 ÷ サイクル時間

一般には，アクセス時間はサイクル時間の約半分である．帯域はアクセス幅とサイクル時間の意味を結びつける最も重要な数である．帯域はメモリが定常的な流れとして情報の授受を行える最大の速度と考えられる．より大きな主記憶装置が必要とされるにしたがい，1つのモジュールで構成すると大きな配列となり，よりながい遅延時間や雑音の増加を招き，適切でなくなる．これらの理由から主記憶装置はしばしば数個のモジュールとして作られる．その結果として複数のモジュールが同時に読み書きのサイクルが行なえ，アクセス時間当りにより，多くのビットを転送でき，それによってCPU から見た主記憶装置の帯域が増大する．記憶装置の語のアドレスは一般に**インタリーブ**される．すなわち，N 個のモジュールに N を法とする方法で番地を割当てる．そのため，アドレス 0, 1, 2, $\cdots$, $N-1$ はすべて異ったモジュールに与えられる．この番地付けにより,引続いて要求されるメモリの語が連続したアドレスにあれば,最大の帯域が得られる．最良で最も簡単な場合には，帯域は単一モジュールの N 倍に増加する．同じモジュールに続けてアクセスするような最悪の場合でも，帯域は単一モジュールの装置と同じである．これらの両極端の中間にも他に簡単な場合がある．すなわち各番地が互いに等しい確率で現われるようなランダムな要求のパターンの場合である．この場合，N モジュールの有効帯域，B_E の概略値は次式で与えられる†．

$$B_E \fallingdotseq B_1\sqrt{N} \qquad (1)$$

† 6章　参考文献 HE 1 参照．

ここで，B_1 は単一モジュールの帯域である．

上記の帯域解析は非常に簡単なものであり，主記憶装置の速度評価の指針としてのみ役立つであろう．複数のモジュールがあり，一定の遅れがある共通の“バス”を持つ場合はより複雑となる． バスの遅れはモジュールに要求を届ける時と，出力が処理装置に送られる時の両方に課せられる．

IBM のシステム/360 モデル 85 やシステム/370 モデル 155, モデル 165 のような多くのシステムでは，大きな“主記憶”へのアクセスは， CPU と主記憶装置の間に置かれた小さな高速**バッファ記憶**に対し，連続する語のブロックとして行なわれる．主記憶はインタリーブされるので，このようなアクセスは主記憶装置を最高速で使うことになる．この型のシステムの詳細は第10章に示す．

6-2 CPU 命令 [AP, BE, GI, RA, WH, WI 参照]

CPU 速度を特徴づける主な2つの方法は (1) カーネル，(2) 命令ミックスである．最初の方法は**カーネル**，すなわち CPU が高率で動作すると考えられる小さなプログラム・セグメント (たとえば， 密に構成されたプログラムの1つ，または複数個の内部ループ) をアナリストが選ぶか，書くことから始める．選ばれたプログラムの構造はかなり簡単であるから，アナリストはプログラムの論理をたどってカーネル実行時の各命令タイプの頻度を計算できる．この頻度ベクトルに計算機の命令実行時間ベクトルをかけて，カーネルの全実行時間が得られる．このような時間は種々のマシンおよびマシンの特徴に対し定められ，比較される．この方法は簡単に使えて短時間で結果が得られる．しかし，選ばれたカーネルがどれほどシステムの活動を代表するものであるかという疑問や，カーネルがそれをコーディングする人のプログラミングの巧妙さと流儀に左右されるなど，いくつかの欠点がある．カーネルによる解析は CPU 設計の初期段階に CPU の性能を短時間で知るために用いられる．

命令ミックス法は(本質的に)簡単な統計平均の手法である．まず，代表的と考えられるプログラムやプログラム・セグメントの集合を選ぶ．その集合の各プログラムは 特別なトレース・プログラムによって[†] **インタプリティブ**に実行される．そのトレース・プログラムは各実行命令をとりそのタイプを判別し，そしてその命令タイプの頻度カウンタを更新する．トレースが行なわれた後に

† 命令実行の統計をとるハードウェア・モニタ(4-5節参照)を 用いることもできる． 各命令のタイプはモニタのカウンタの1つに割当てられる． そして命令実行の際，マシンの回路から発せられるパルスが加えられる．この方法は上述のソフトウェア・インタプリタほど用いられない．

命令の相対頻度(図6-2.1参照)に命令実行時間がかけられる．そして，その和が(トレースされたプログラムの)平均CPU実行時間である．

まったく同じか，極めて類似した命令セットを持つが構成要素の速度が異なる計算機を比較するのには，同じ頻度ベクトル(時間ベクトルは異なるが)が両方の命令セットに用いられるので，ミックス法は大変有効である．同じ頻度ベクトルを用いるミックス法はアーキテクチャ(命令セット，レジスタ等)の異なるシステムを比較するのには適さない．この場合でさえも，同じソース・プログラムがおのおののトレースのソースとして用いられるならば，平均命令時間の有効な比較が可能である．しかしアナリストは微妙な要素が結果に影響を与えることを知っていなければならない．たとえば2つのトレースされる目的プログラムを生成するコンパイラの目的コードの効率が違うことによる影響がある．

図6-2.1は引用文献から再掲したあるミックスを用いる全計算である．頻度

命令のタイプ	相対頻度	命令実行時間	重みつき時間
ロード/ストア	0.475	4.4	2.090
固定小数点　加算/減算	0.080	4.4	0.352
インクリメント　1	0.030	13.0*	0.390
固定小数点　乗算/除算	0.003	9.3	0.028
浮動小数点　加算/減算	0.005	15.3	0.077
浮動小数点　乗算/除算	0.002	10.0	0.020
AND/OR	0.070	4.4	0.308
分岐(無条件，レジスタ)	0.125	2.2	0.275
分岐(インデックス修飾)	5.035	4.4	0.154
テスト(レジスタと記憶装置)	0.040	4.4	0.176
テスト(レジスタとゼロ)	0.025	4.4	0.110
その他のテスト	0.005	4.4	0.022
シフト	0.075	7.6	0.570
その他	0.030	4.4	0.132
平均実行時間			4.704

*1命令以上

マシン：　IBM　7090(2つの演算レジスタ，3つのインデックスレジスタ)

問　題：　"コマンド・コントロール問題によく似たアプリケーション"

参　照：　E. Raichelson and G. Collins: AMethod For Comparing the Internal Operating Speeds of Computers, Comm. ACM, vol. 7, no. 5, May, 1964.

図6-2.1　命令ミックスの計算を示す表

を与える問題の仕様があいまいな点に注意しよう．このようなあいまいさはこれらのレポートでは典型的である．命令ミックスの他のレポートでも，かなり異なった問題についてさえ，ロード/ストア命令が非常に大きな割合を示めるとされている．このことは IBM 7090 のようにレジスタの数が少ないマシンでは容易に理解できる．命令頻度のあるものは問題によって極めて異なることがある．たとえばコンパイルのみを実行して得られるミックスでは浮動小数点演算は非常に稀であるが，他の(“科学的な” 応用の) 目的プログラムのミックスでは，浮動小数点演算は実行される命令数の 17% にもなる．(そして，実行時間ではもう少し高い割合となる.)

ミックス法は計算機のある特徴の価値をよく理解するのに役立つ．**図 6-2.2** に同じバブル・ソート問題で 2 進固定小数点数の 3 つの計算機をミックス解析した結果を示す．すべてのマシンは同じ命令セットを持っているが，レジスタ

命　令	時間/命令	8 レジスタ，インデックス修飾あり		2 レジンタ，インデックス修飾なし*		2 レジスタ，インデックス修飾なし	
		相対頻度	時間割合	相対頻度	時間件合	相対頻度	時間割合
加　算	2.0	0.199	0.214	0.157	0.161	0.101	0.104
減　算	2.0	0.013	0.013	0.014	0.014	0.017	0.018
乗　算	4.0	0.000	0.000	0.000	0.000	0.000	0.000
除　算	6.0	0.000	0.000	0.000	0.000	0.000	0.000
シフト	1.0	0.018	0.010	0.014	0.007	0.013	0.006
ロード	2.0	0.223	0.239	0.304	0.312	0.310	0.319
ストア	2.0	0.201	0.215	0.304	0.312	0.304	0.314
比　較	2.0	0.120	0.129	0.090	0.094	0.046	0.047
AND, OR, EXOR	2.0	0.000	0.000	0.000	0.000	0.121	0.125
	1.0	0.219	0.118	0.111	0.052	0.085	0.044
入出力	21.0	0.005	0.051	0.003	0.036	0.002	0.005
平均命令時間			1.854		1.936		1.938
実行命令数			13087		17310		37717
実行時間			24263		33512		73096

*間接アドレス方式を使用

問題：　80 欄の 14 枚のカードをソートせよ

キーフィールド：3-11 欄

アルゴリズム：バブル・ソート

図 6-2.2　簡単なソーティングの命令頻度

の数や間接アドレス指定が許されているかどうか等の違いがある.

大型機, 特にCPUに命令スタック, バッファ記憶, パイプラインが用いられている計算機に対しては, CPU性能を測定する際, 記憶装置参照のパターンが命令コードの頻度とおそらく少なくとも同程度に重要である. ミックス法はこの要素を考慮できない.

命令ミックス法やカーネル法はバッファ記憶を持たないCPU構成の性能評価に役立つ手法であるが, 全システムの性能は, そのシステム性能がCPUによって強く制限されているような特別の場合を除いて評価できない.

最後に, ミックス法に必須な, 命令をトレースするプログラムを書くにはかなりの努力が必要とされることを付加えておく. これは計算機毎に一度だけ書く必要があり, 命令にEXECUTE(代行)タイプを持つ計算機ならばより簡単である. そのようなインタプリタはトレースされるプログラムより実行が非常に遅い. (一般に20~30倍である.) このため命令ミックスの生成は計算機時間を犠牲にして得られる.

6-3　マイクロプログラム制御 [HE, HU, RO, TU, W1参照]

ただ1つのマシン命令を実行するにも, 計算機の内部回路のいろいろな経路に, 命令自体とその命令が参照するデータの両方の情報を流すスイッチング操作の系列が必要とされる. そのような系列を**マイクロプログラム**と呼ぶ. これは計算機設計に基本的な概念であり, 1950年代にM. V. Wilkesによって提唱された†. しかし, 1960年項までは多くのマシン設計者は彼等の仕事や計算機制御の構成が一種のプログラミングを含むとは考えなかった.

1960年代中項から, マイクロプログラミングはCPUやI/O制御回路の設計法として広く使われるようになった.

マイクロプログラミング構成では, スイッチング制御情報は**制御記憶**(control store: 略して**CS**)と呼ばれる記憶装置に収められるマイクロ命令に編成される. CSは計算機の主記憶より一般に数倍速い. 経済的理由から, しばしば**読出し専用記憶**(read-only store: 略して**ROS**)が用いられる. これは電子的速度で高速に読出せ, その内容を変更するには低速の機械的操作を必要とする. 単一のクロック回路がマイクロ命令を1つづつCSからCSデータ・レジスタに送る. そこからマイクロ命令の各ビットはデータフロー構造の, すなわち, メモリ・アドレス・レジスタ, メモリ・データ・レジスタ, 加算器, シフタ, プログラム・カウンタおよびいろいろな計算機レジスタの間の, 情報を制御す

† 参考文献W1参照.

る信号を与える．各マイクロ命令は上述の制御情報を与えるだけでなく，次のマイクロ命令の格納場所を部分的に決めるビットも与える．その場所はレジスタの符号ビット，桁あふれなどの状態を示す選ばれたフリップ-フロップの状態によっても決められる．（マイクロ命令分岐）マイクロプログラミングについて詳細に述べることはこの本の範囲を越えている．しかし，初めにもう1つの方式の配線による計算機制御と比較してマイクロプログラミングが有利な主な点を列挙してから，幾つかの性能問題を検討することにしよう．

1. **経済的な設計と保守の容易性**　マイクロプログラムの論理設計の多くはプログラミングの仕方に似ており，シミュレーションやコンピュータ援用を含めて規律正しい方法を用いることができるのが有利である．制御系列は極めて定式化されているので，計算機の保守，変更がより簡単である．
2. **ハードウェアの経済性**　制御記憶の費用を無視できるような複雑な命令セットを考えれば明らかである．
3. **エミュレーションによる互換性**　与えられた計算機に純粋なソフトウェア・インタプリタによるよりはるかに高速に，他の計算機のすべての命令セットを“まね”させることができる．これにはマイクロプログラムの幾つかのセットをCSに入れればよい．こうして同じ計算機が“新しい”命令セットを持ち，新しいオペレーティング・システムを走らせ，（他の命令セットを用いて）古い型の計算機から利用できる多くのプログラムをもってきて走らせることができる．
4. **特殊機能や特殊化**　特殊機能，あるいは応用プログラム全体でさえもマイクロプログラム化されるが，そのときは同じ計算機でプログラムした同じ機能よりずっと速く走る．

上記の第2と第3の点が現在マイクロプログラム計算機制御の用いられる主な理由である．広い範囲の型について同じ複雑な命令セットを持っているIBMのS/360やS/370システムはその最も著しい例である．

特殊な目的に特注のマイクロプログラムを使うことについて，たくさん提案され議論されてきた．パフォマンス改善の可能性は，マイクロ命令が実際の計算機資源を**直接に**アクセスする制御“原子”であるという点にある．これに反して，（マイクロプラグラム・シーケンスで作られる）通常の命令は，計算機資源を使用するシーケンス全体を与える制御“分子”である．どのような特別な機能もそれ自身制御の分子であるから，原子，すなわちマイクロ命令でこの分子を作るのはあらかじめ作られた他の分子より効果的である．特に次の点がマイクロプログラムによる速度の改善の主な原因であろう．

1. 命令フェッチの減少．これはマイクロ命令フェッチにとって代られる．より高速な記憶装置(主記憶装置でなくCS)から行なわれ，特定の機能に合せて特殊になっている．
2. 注文された目的に対するハードウェア・レジスタ，高速記憶装置等の，より効率的使用

ある特定の機能をマイクロプログラム(プログラムではなく)化することを正当とするためには要求された機能の予期される使用頻度，そして，良いマイクロプログラムを書くのに必要なかなりの努力と手腕などの要素を考慮することが重要である．マイクロプログラム化する機能を選択する合理的な方法の1つは，システムの動作負荷(第4.2節参照)として典型的であるとわかっているか，そう見込まれるジョブを1つ以上選び，そしてトレース・プログラムのもとでそのジョブを走らせることである．トレースによって実行された各命令の命令コードが少くとも記録される(たとえば，磁気テープ上に)であろう．命令操作の流れを解析し，よく現われる部分系列を見つける．この部分系列はマイクロプログラムで実現するときの最初の候補である．そのような機能に対するマイクロプログラムの荒設計は測定したプログラムのトレースの流れの中の系列と比較される．この両者の時間は，その機能の流れの中であらわれる頻度によって重みづけされると，マイクロプログラムによる実現がパフォマンス上に与える利益の実際的な査定がなされる．パフォマンス改善をもっと内輪に見積るにはマイクロプログラムを，注意深く改訂した“最上の”プログラムに対して比較すべきである．今日まで，そのような比較についてほとんど発表されていない．直感的に，ある環境の下でのマイクロプログラムで実現されるべき最初の候補は特別の数学的な関数(平方根，対数，指数，三角関数など)と考えられる．これらの関数の多くは，多項式として扱われ，その係数を高速のレジスタに蓄えることで利益を得ることができよう．もう1つ直感的な感じでは，プログラミング言語が型の異なる変数の演算をプログラマに許すようになって来ているので，データ変換操作(固定小数点から浮動小数点へなど)もまたかなりのCPU時間を消費しているであろう．

汎用のシステムを供給しているコンピュータ製造者の多くはユーザにユーザ自身のマイクロ命令の集合を書くことを奨めいてない．もしそうすると，いやに凝ったオペレーティング・システムやコンパイラを含めて，高価なソフトウェアの費用は広く一般のユーザに負担させるものであるから，それらを提供することが大変困難となるであろう．性能の点からみると，ひどく異なった命令の集合(ただ異なったマイクロ系列によるだけだが)を持つ2つの同一のハード

ウェア・システムは(ハードウェアは同一だけれど)まったく異なった計算機である．それらの比較には，任意の2つの異なった計算機システムを比較すると同じ程度の複雑さとあいまいさがある．

6-4 I/O と CPU のオーバラップモデル†

最近のシステムの多くでは，I/O 機器［テープ，ディスク，ドラムなどの**補助記憶装置** (AS: auxiliary storage) を含めて］は CPU や**主記憶装置** (MS: main storage) からほぼ独立して動作する．このことはこれらのすべてが**並列的に**動作可能である，すなわち，それらの使用は時間的にオーバラップされうることを意味する．そのようなオーバラップには多くの場合がある．この節では，CPU を1つの装置とみなし，その使用と I/O 装置の使用のオーバラップを，高度に理想化した動作負荷とシステムのモデルを用いて考察する．

7章では，各装置が同時に異なったジョブの異なった部分の仕事するような SPOOLing と呼ばれるオーバラップの種類について議論する．ここで考えられる事例はこの種の重複によく合っているかも知れないが，同一ジョブの各部分に重複が見られる場合に，もっと素直にあてはまる．

図 6-4.1 は CPU と主記憶，および(高々) 2 つのデータ・チャネルを介して主記憶装置に接続されている2つの装置からなるリソース構造を示している．実際のシステムでは，データ・チャネルは I/O 機器と主記憶の間の物理的な

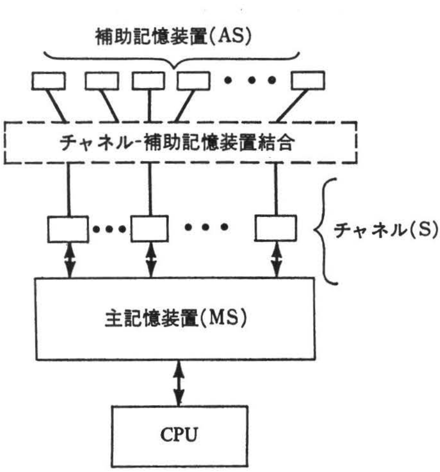


図 6-4.1 リソース構造

† この節の多くは参考文献 HE 2 に依っている．

信号線と，それに伴なう制御，バッファ回路から成り立つ．これにより I/O 転送や信号制御が CPU と独立して動作しうる．1つのチャネルに数個の装置が物理的に接続されていても，1つしかないチャネルの制御回路はその装置の中で高々1つの装置しか同時には動作させないことを意味する．われわれのモデルでは，それゆえ，チャネルの数は同時に動作できる I/O 装置の数とする．

動作負荷は1つの装置に収められた**問題レコード** (problem record) の一連の集まりとしてモデル化される．各問題レコードは一続きのバイトからなる．MS と AS の間で s バイトの列をアクセスして，転送するのに要する時間は次式で表わされる．

$$T = t_a + k_f s \tag{1}$$

ただし，

t_a = 列の最初をアクセスする時間
s = 列中のバイトの数
k_f = バイト転送速度の逆数

もちろん，実際はずっと複雑なプロセスであり，式(1)はそれを簡略にしたものである．たとえば，t_a はディスク・ファイルでは定数ではなく，新しい物理レコードと前の物理レコードに対する相対的位置に依存する．また，ディスク装置ではバイト数がシリンダの容量を超えたら，再度アクセスが必要となるので，s は余り大きくはできない．式(1)はディスクよりテープに近いモデルである．t_a は相当大きいので，問題レコードを**ブロック化**，すなわち，各転送に数個の問題レコードを含めて s をかなり大きくし，それによって各 t_a で数個の問題レコードを取扱うことがよく行なわれる．レコード・ギャップの数が少なくなるので，ブロッキングはまたテープ/ディスク記憶装置を有効利用する．ブロック化係数 n は次の様に定義される．

$$n = \lceil \frac{s}{r} \rceil \tag{2}$$

ただし　s：1ブロックのバイト数
r：1問題レコードのバイト数
$\lceil\Gamma\rceil$："下回らない最小の整数"

われわれの理想化されたモデルでは，レコードの各ブロックは3つのフェーズのプロセスを受け，各フェーズの継続時間はすべてのレコードについて同じであるとする．一般的なブロック i に対する各フェーズの時間は次のように表わされる．

$I = I_i =$ ブロック i を入力する時間(アクセス時間 + 転送時間)
$C = C_i =$ ブロック i を演算する時間
$O = O_i =$ ブロック i を出力する時間(アクセス時間 + 転送時間)

動作負荷(ジョブ)は全部で N 個の問題レコードから成っており，N は充分大きく，ジョブをスタートさせたり終了させたりする時間は定常的なレコード処理時間に比して無視できるとする．動作負荷モデルに対するその他の仮定は以下のように要約できる．

1. レコードはすべて同じ大きさである．(等しいバイト数)
2. 各入力レコードに対し，1 つの出力レコードがある．
3. ある 1 つのオーバラップの場合では，ブロック当りのレコード数は全ブロックについて等しい．
4. あるレコードに対する CPU 計算時間は各レコードについて等しい．
5. ある 1 つのオーバラップの場合では，装置のアクセス，転送時間はすべてのレコード・ブロックについて等しい．

4 つのシステム構成を考え，上記の動作負荷に対する比較を行なう．それらはチャネル数が異なり，それゆえ，オーバラップ可能度や，主記憶装置の分割法が異なる． しかし，主記憶装置の全容量は異ならず，4 つの構成とも M 個の問題レコードがはいる．**図 6-4.2** に 4 つの構成の主記憶装置の分割とオーバ

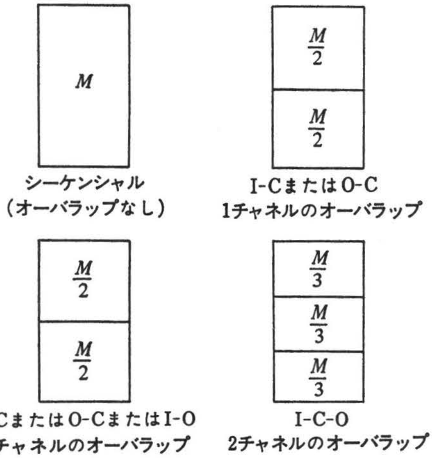


図 6-4.2 主記憶装置の分割

ラップ可能な型を示す.

1. S(シーケンシャル)——オーバラップなし. 入力, 出力, 演算は完全に順に現われる. すなわち, どれかが実行されているときは, 他の2つは実行されない. ($p=1$)
2. I-C, O-C——入力と演算, または出力と演算は同時に行なえる. ($p=2$)
3. I-C, O-C, I-O——入力, 演算, 出力の3つのフェーズの内の2つが同時行なえる. ($p=2$)
4. I-C-O——入力, 出力, 演算がすべて同時に行なえる. ($p=3$)

各場合の p の値は単にプロセス中で同時に存在し得るフェーズの最大数であり, それは第1-3節の証明によれば直列的なシステムでのスループット改善度の上限である. M 個の問題レコードを収容できる主記憶装置は, バッファ

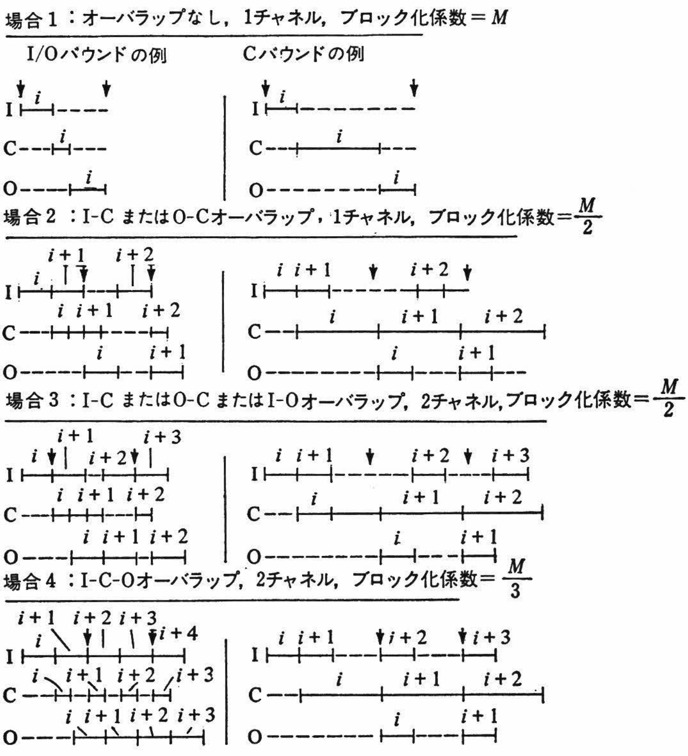


図 6-4.3　タイミング・シーケンスの例

用と演算用に p 個の等しい領域に分割される.（これにはプログラムやスーパバイザ等のための場所は勘定していない.）各並列フェーズはそれ自身の MS 領域を持たなければならない.

種々のオーバラップの場合を判りやすくするため，**図 6-4.3** に 4 つ各々の構成のタイミング・シーケンスの例を 2 組づつ示す．この例では

（i） 演算 ＝ 1 時間単位，I/O ＝ 2 時間単位

（ii） 演算 ＝ 5 時間単位，I/O ＝ 2 時間単位

(i)は I/O が制限となる条件に，(ii)は演算が制限となる条件に対応する.

オーバラップが許されていない最も簡単なシステムでは，プロセス・ブロック i の合計時間は単に各フェーズ時間の合計である.

$$t_{B_i} = \mathrm{I}_i + \mathrm{C}_i + \mathrm{O}_i \qquad (3)$$

次に I-C または O-C オーバラップを許す 1 つのデータ・チャネルを持ち高速化されうるシステムを考察しよう．最初の場合として，レコード i をシステムが演算している間に，レコード $i+1$ が入力フェーズであるか，レコード $i-1$ が出力フェーズの場合を考える．この場合

$$t_{B_i} = \mathrm{Max}(\mathrm{C}_i,\ (\mathrm{O}_{i-1} + \mathrm{I}_{i+1})) \qquad (4)$$

今後われわれは各フェーズの時間は i に無関係で，すべての i について $\mathrm{C} = \mathrm{C}_i$, $\mathrm{I} = \mathrm{I}_i$, $\mathrm{O} = \mathrm{O}_i$ と仮定する．このとき式(3)(4)は次のようになる.

$$t_B = \mathrm{I} + \mathrm{C} + \mathrm{O} \qquad \text{(直列)} \qquad (5)$$

$$t_B = \mathrm{Max}(\mathrm{C},\ (\mathrm{I} + \mathrm{O})) \qquad \text{(I-C または O-C オーバラップ)} \qquad (6)$$

2 つ目のチャネルを導入すれば，更に幾つかのオーバラップの可能性がある．たとえ 2 つのフェーズがオーバラップするとしても，演算と，入力または出力がオーバラップでき，あるいは入力が出力とオーバラップできる．それらの中には 2 つの場合がある．まず，$\mathrm{C} \geqslant \mathrm{I} + \mathrm{O}$ の場合，入出力の 1 つのサイクル全部が演算と並行してなされるので処理時間が重要な要素となる．次に，$\mathrm{C} < \mathrm{I} + \mathrm{O}$ の場合，2 つのサイクル全体を考慮しなければらなない．この場合，$\mathrm{I} + \mathrm{O}$ の C より多い部分は他の入力または出力動作とオーバラップでき，それで完全なオーバラップが達成される．これは次のように表現される.

$$t_B = 1/2(\mathrm{C} + \mathrm{Max}(\mathrm{C},\ \mathrm{I} + \mathrm{O})) \qquad \text{(I-C, O-C, I-O オーバラップ)} \qquad (7)$$

2 チャネル構成では，並行に 3 つまでの動作が許される．そこでブロック当り

の処理時間を与える式は次のようになる．

$$t_B = \mathrm{Max}(\mathrm{C} + 1/2(\mathrm{I} + \mathrm{O})) \qquad (\text{I-C-O オーバラップ}) \tag{8}$$

さて，これらの基礎的な関係(読者は図**6-4.3**の例の場合で確かめられたい)をユーザが興味のある問題で知っていたり，あるいは，選択したりすると思われる**基本変数**(primary variable)を用いて書き換えよう．その変数は

$t_c =$ 1つの問題レコードの演算時間
$t_a =$ レコードの1つのブロックの転送を開始するためのアクセス時間
$t_f =$ 1つの問題レコードを転送する時間
$M =$ 主記憶装置の容量，ワーク・スペースやバッファ用として使用できるレコードの数で表わす．
$n =$ ブロックあたりのレコード数(ブロック化係数)
(すべての時間はミリ秒で示す)

これらの変数は式(5)(6)(7)(8)の I, O, C に次のように関係づけられる [I/O については式(1)を用いる]

$$\mathrm{C} = nt_c \tag{9}$$

$$\mathrm{I} = t_a + nt_f \tag{10}$$

$$\mathrm{O} = t_a + nt_f \tag{11}$$

式は基本変数を用いて表わしたが，直接ユーザに関係のある補助変数が3つある．

$r =$ レコードの大きさ(バイト数で示す)
$m =$ バッファやワーク・スペースとして用いうる主記憶装置のバイト総数
$k_f =$ 補助記憶装置の転送速度(キロバイトで示す)

補助変数と基本変数は次式で関係づけられる．

$$t_f = \frac{r}{k_f} \tag{12}$$

$$M = p\left\lfloor \frac{m}{pr} \right\rfloor (\text{上まわらない最大の整数}) \tag{13}$$

オーバラップの構成の比較を簡単にするため，この式を次のように少し変更して用いる．

$$M = L\left\lfloor \frac{m}{Lr} \right\rfloor \tag{13-a}$$

	オーバラップ	チャネル数	ブロック化係数	レコードあたりの経過時間 一般式	詳細化	正規化
1	シーケンシャル (オーバラップなし)	0	M	$t_B = I + C + O$	$t = t_c + \frac{2t_a}{M} + 2t_f$	$T = 1 + 2A + 2F$
2	I-C,O-C オーバラップ	1	$\frac{M}{2}$	$t_B = \mathrm{Max}(C, I + O)$	$t = \mathrm{Max}\left(t_c, \frac{4t_a}{M} + 2t_f\right)$	$T = \mathrm{Max}(1, 4A + 2F)$
3	I-C,O-C,I-O オーバラップ	2	$\frac{M}{2}$	$t_B = ½(C + \mathrm{Max}(C, I + O))$	$t = \frac{t_c}{2} + ½\mathrm{Max}\left(t_c, \frac{4t_a}{M} + 2t_f\right)$	$T = ½ + ½\mathrm{Max}(1, 4A + 2F)$
4	I-C-O オーバラップ	2	$\frac{M}{3}$	$t_B = \mathrm{Max}(C, ½(I + O))$	$t = \mathrm{Max}\left(t_c, \frac{3t_a}{M} + t_f\right)$	$T = \mathrm{Max}(1, 3A + F)$

場合	条件	Cバウンド 経過時間/レコード 詳細化	正規化	I/Oバウンド 経過時間/レコード 詳細化	正規化
1	—	$t = t_c + \frac{2t_a}{M} + 2t_f$	$T = 1 + 2A + 2F$	$t = t_c + \frac{2t_a}{M} + 2t_f$	$T = 1 + 2A + 2F$
2	$t_c > \frac{4t_a}{M} + 2t_f$	$t = t_c$	$T = 1$	$t = \frac{4t_a}{M} + 2t_f$	$T = 4A + 2F$
3	$t_c > \frac{4t_a}{M} + 2t_f$	$t = t_c$	$T = 1$	$t = ½\left(t_c + \frac{4t_a}{M} + 2t_f\right)$	$T = ½ + 2A + F$
4	$t_c > \frac{3t_a}{M} + t_f$	$t = t_c$	$T = 1$	$t = \left(\frac{3t_a}{M} + t_f\right)$	$T = 3A + F$

記　法

r ＝問題レコードあたりのバイト数
t_c＝問題レコードあたりの計算機時間 単位ミリ秒
m ＝バッファ,動作スペースのための計算機記憶容量 単位バイト
$M = [m/r]$＝問題レコードの数 m
t_a＝問題レコードの一つのブロックにアクセスする補助記憶のアクセス時間 単位ミリ秒
k_f＝補助記憶装置のデータ転送速度　キロバイト/秒
$t_f = r/k_f$＝一つの問題レコードの転送時間 単位ミリ秒
$A = t_a/Mt_c$＝正規化されたアクセス時間
$F = t_f/t_c$＝正規化された転送時間
$T = t/t_c$＝正規化された合計時間
ブロック化係数＝1回のアクセスで読み/書きされる問題のレコードの数＝n

図 6-4.4　レコードあたりの経過時間の要約

ここで，L は種々の場合に対する p の最小公倍数である．$p(1)=1$，$p(2)=2$，$p(3)=3$ であるから，$L=6$ である．

変数 t_a は装置によって厳密に定まる．レコード・サイズが前もって決められていると仮定すれば，t_f は補助記憶装置の物理的特性で決まり，M は主記憶装置の容量とブロック化係数で決まる．また t_c はプロセッサ速度と問題によって決まる．

図 6-4.4 には各構成についてのレコード当りの経過時間の式，要求チャネル数，ブロック化係数（1度のアクセスと転送操作で動く問題レコードの数）を示す．使用可能スペースは作業（計算）スペースやバッファ用に同じ大きさの部分に分割する．各領域（レコード中の）大きさはブロック化係数に等しい．またCバウンド，I/O バウンド† の状況を与える不等式も示す．

レコード当りの経過時間は逐次処理にとっては大変簡単な概念であるが，オーバラップを許す場合は余り明白ではない．その時間はすべてのケースとも n 個のレコード全部を処理（すなわち，計算と両方向の転送）する合計時間を表わす式を書き，それを n で割って導びかれる．

式を調べると逐次処理のとき，ブロック化係数は改善できるが，これはアクセス間 (t_a) が支配的であるとき，とくに重要であることが示される．ブロックあたりのレコード数がより大きいとアクセス時間による損失はより少なくなるので，直列処理は2チャネル I-C-O オーバラップ構成より良い性能を与える．

簡単な逐次処理での最良の改善度は

$$\frac{3t_a}{M} \leqslant t_f \tag{14}$$

$$t_c = t_f \tag{15}$$

となる I-C-O 構成で得られる係数3であることは興味がもたれる．

レコード・サイズ，主記憶装置サイズ，補助記憶装置が指定されたら，設計者の決定で式(14)の成立を左右することはできない．式(14)が成立したら，式(15)は（レコードあたりの計算時間）t_c を調整することによってのみ満たされる．しかし，t_c は問題と CPU 速度の両方に影響される．

図 6-4.5 では処理時間軸上での臨界の間隔を定義し，各オーバラップの場合のスピードを順に並べた．これから，2チャネル・システム（構成3または4）が，必ずしも I-C-O がすべてオーバラップするモードを用いないにせよ，常に最良のパーフォマンスを与えるように構成できることが分る．しかし，もし

---

† C バウンド (compute bound)：計算能力によるスループットの制限
　I/O バウンド (input output bound)：入出力能力によるスループットの制限

処理時間の制限	レコード当りの時間 短 ←→ 長				備　　考
$0 \leqslant t_c < \frac{t_a}{M} - t_f$	3	1	4	2	
$\frac{t_a}{M} - t_f \leqslant t_c < 2\frac{t_a}{M}$	3	4	1	2	もし $t_c = \frac{t_a}{M} - t_f$ なら4=1
$2\frac{t_a}{M} \leqslant t_c < \frac{4t_a}{M} + t_f$	4	3	2	1	もし $t_c = \frac{2t_a}{M}$ なら1=2,3=4 $t_c > \frac{3t_a}{M} + t_f$ に対して 4はCバウンドである
$\frac{4t_a}{M} + 2t_f \leqslant t_c$	4 = 3 = 2				2,3,4 Cバウンド

理想化された場合の相対効率

1 =シーケンシャル：2 = I-C,O-Cオーバラップ：3 = I-C,O-C,I-Oオーバラップ：4 = I-C-Oオーバラップ

図 6-4.5 各場合の比較

システムが計算時間で強く制限されるなら，1チャネル・システムも同程度の能力をもつ．また，1チャネル・システムは入出力で強く制限されるとき，より大きなブロック化係数で純粋な逐次システムとして最もうまく動作することが分る．**図 6-4.6** には図 6-4.4 と図 6-4.5 の式で与えられる臨界条件をグラフ的に示す．直列構成はプログラムも特徴抽出も最も容易なので，他の構成との比較の“標準”として用いるのが便利である．

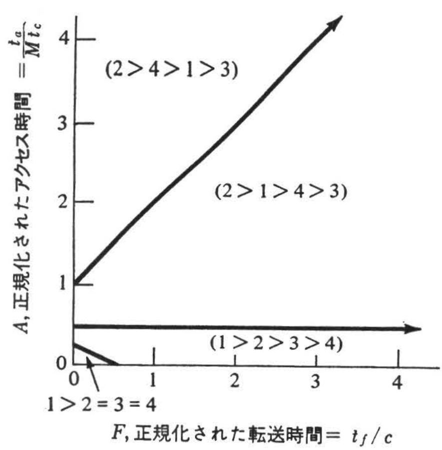


図 6-4.6 各場合のグラフ的比較

最後に p 個のリソースを同時に動作させうるシステムは，同じリソースを直列にしか動作できないシステム $(p=1)$ の最大 p 倍のスループットが期待できる．この理論的改善はある動作負荷で，しかも，システム構成と問題のパラメータを入念に調整したときのみ得られる．他の動作負荷に対しても，装置パラメータとデータ転送プログラムを注意深く選択することにより，改善度は低いが，それでも顕著なオーバラップ(そして高いスループット)が達成される．

これらの式は最良のスループットを与えるキー・パラメータとあるクラスの動作負荷との関係を明らかにしている．この動作負荷は入力，出力，計算のためのリソースをきまって要求する固定長のレコードで特徴づけられる．高度のオーバラップのために求められる主記憶装置の分割は，特にアクセス時間が計算時間の半分程度のとき，スループットに顕著な影響を与えることを，これらの式は示している．2チャネルのシステムは，逐次または1チャネルのシステムより大きなスループットを持つように常に設計可能であるが，その最大値は必ずしもオーバラップが最大のときではない．

6-5　装置の性能のまとめ：IBM システム/360 [AL, GR, IN, OP, RU 参照]

本章で示されたように，計算機装置はいくつかのサブシステム——主記憶モジュール，CPU，補助記憶装置——から成ると考えられる．その他にもプリンタ，カード・リーダ・パンチ，ターミナル，CRT ディスプレイ装置等がある．これら各サブシステムの性能を規定する手法のいくつかについて論じた．そして簡単な場合についてサブシステム間の関係を説明する単純なモデルを，オーバラップ条件を理想化して第6-4節に述べた．全体を通して，動作負荷を考慮しない最も初歩的な“生の”性能要素についてのみ述べられている．システムが一緒に動作する部分の簡単な組合せでさえ，その性能を知るには動作負荷の詳細なデータを要求するが，その性能はその動作負荷に対してだけのものである．このことはコンピュータの機能が極めて普遍的なものであることと，また，システムの構成部分の間の結合にはまったく柔軟性がありプログラムで制御されることを思うと，驚くにはあたらない．

図6-5.1は動作負荷に関係ないと考えられるハードウェア性能の要素を簡潔にまとめたものである．この形式は完全な摘要書を意図したのではなく，与えられたシステムのハードウェア能力性能の性能像を得るための手始めである．あるシステムについて，表の各項目の実際の値を得ることのむずかしさを読者自身で発見して載きたい．ある情報は発表されていないし，また，各メーカ固

機種名________ モデル________ 製造会社________
発表年月日________ 参考資料________ 日付________
主記憶容量________ サイクル・タイム________ ビット/アクセス________
インタリーブ係数________ 最大帯域________
チェック・ビット数/アクセス________
命令数________ ビット/命令________ 命令数/アクセス________
CPU アドレス空間________ 間接アドレス方式________
インデックス・レジスタ数________ アドレス可能なレジスタ数________
CPU バッファ：ビット数________ 常域________
リアル・タイム・クロック________ 再配置レジスタ________ 保護機構________
割込みロケーション数________ インタラプト最小時間________
ビット数：F. P. 語________ *F*1. Pt. キャラクタ________ *F*1. Pt. 分割________
固定小数点タイミング：　加算________ 比較________ 分岐________ 乗算________
浮動小数点タイミング：　加算________ 比較________ 分岐________ 乗算________
256 バイト転送時間(主記憶-主記憶)________
I/O サブシステム：
アドレス可能な装置数________ 並列転送数________
バイト形式：情報ビット数________ チェック・ビット数________
最大 I/O バイト・レート________ % CPU 干渉________
第一補助記憶装置：タイプ________ 容量________
平均アクセス________ バイト・レート________
バイト数/アクセス________
他の装置，備考：________

注：時間，帯域，記憶容量の単位も含めること

図 6-5.1 マシン概要

有の用語を使っていて表の分類に対応させるのが困難なことがわかるであろう．また，よく考えると非常に複雑であるが，一見細かなことが大きな意味を持ちうる．たとえば，固定小数点加算の時間でさえも，ある加算命令は 2 つの高速レジスタからオペランドを取り(IBM システム/360 RR ADD のように)，一方，他の加算命令は 1 つのオペランドを低速の主記憶装置から取りだす(RX ADD のように)ので，同一システムでいくつかの値を持つのも当然である．また，高速な命令スタック-バッファ-キャッシュを持つマシンでは，命令やデータがその高速な領域にあるかどうかで，加算時間が非常に異なる．これらの理由から，図 6-5.1 はシステムの比較に際して充分な注意を払って用いられる

べきである．これらの難しい問題がある場合に，確固たるパフォーマンス像を得ることは動作負荷の統計的性質，特にその記憶装置参照のパターンに依存すると考えられる．現在のところ，これらの性質を簡潔に分類するうまい方法はない．

IBM システム/360 計算機系列は機能的に同じモデルの集まりとして計画された．充分なシステム・リソース，特に充分大きな主記憶装置が使用可能ならば，それらはすべて同じ機械語のプログラムを走らせることができる．各モデルは速さ，価格共に非常に異なる．速さの比較を**図 6-5.2** に示す．（この図表は参考文献 HE 1．による）．"内部相対速度" はミックス解析から得られたものである．あるモデルと次のモデルとでは速度の比は約3倍であり，速度増加の主な理由は主記憶装置の幅とインタリービングであることがわかる．いろいろなシステム構成要素の速度と内部速度を正確に関係づけることは非常に難しいことに注意しよう．

6-6　コンパイラ性能

ほとんどすべての計算機使用者は現在，一般的に**ソース言語**(source langu-

モデル	相対内部速度	主記憶装置				制御		汎用レジスタと浮動小数点レジスタ			加算器巾	回路遅延
		サイクル	幅	インタリーブ	帯域	タイプ	サイクル	タイプ	幅	サイクル		
30	1	2.0	1	—	500	ROS	1.0	core	1	2.0	1	0.030
40	3.5	2.5	2	—	800	ROS	0.625	core	2	1.25	1	0.030
50	10	2.0	4	—	2,000	ROS	0.5	core	4	0.5	4	0.030
65	30	0.75	8	2	14,933	ROS	0.2	circ	4	—	7	0.010
75	45	0.75	8	2-4	21,300	circ	0.195	circ	4.8	—	8	0.006
85	120	1.04	16	2-4	N. A.	circ	0.80	circ	8	—	8	0.005
91	135-420	0.75	8	8-16	N. A.	circ	0.60	circ	8	—	8	0.002

注

1. 幅はすべてバイトで示す．
2. サイクル遅延時間はすべてミリ秒(ナノ秒にするには 1000 倍せよ)．
3. 記憶装置帯域はインタリーブの平方根の係数がかけられている．
4. ROS は "読み出し専用記憶"，circ は "論理回路" を示す．
5. モデル 85 は(0.080 マイクロ秒アクセスの) 16 または 32 キロバイトの "キャッシュ" メモリを含む．

図 6-5.2　IBM システム/360 の各モデルの比較

age)と呼ばれるシンボリック言語でシステムと通信する．これは機械語またはオブジェクト言語[†]に翻訳されなければならない．この過程での性能に関するものとして次の3つのクラスがある．

1. **言語性能**　ユーザの使い勝手，これは“ユーザ性能”，すなわちプログラムを書き，デバグする時間と解釈される．
2. **トランスレーション性能**　コンパイラ・プログラム(又はインタプリタ)の必要記憶装置と速度
3. **オブジェクトコード性能**　コンパイラにより生成されるオブジェクト・プログラムの必要記憶装置と速度

これら3つの分類は言語とコンパイラの評価の基本的なものである．各々は独自のパラメータで影響をうけるが，各項の相互作用もある．次に述べる簡単な概要では，われわれはその考察と技術の現状について触れる．詳しくは参考文献を参照されたい．

言語性能は大変重要で多分3つの分類の中で最も重要であると，ほとんどすべての人に思われているが，定量化するのは最も難しい．言語の効率の定量的な研究の発表も，そのような結果を得るための方法論もほとんどない．このことは計算機言語が自然言語と同様，われわれの最も基本的な思考過程に密接に結びついていて，その思考過程が心理学者，言語学者やその他の人々の大変な努力にもかかわらず，まだ正確な解析を拒んでいるので，そんなに驚くべきことではない．測定の観点から，最も簡単な記述は多分ソース・プログラム中の文字の数である．これと2~3の他の記述子が，2つの言語を比較するための次のような実験に含まれるであろう．(1) 解くべき問題とデータ例を選んで集める．そして，(2) 2つの言語でそれらをコード化しデバグするプログラマのグループに問題を与える．彼等が問題を解く努力をしている間プログラムを書きデバグする時間と，ソース・プログラムの長さ(文字数)を測定する．少し考えるとこの手続きには多くの落し穴があることに気付くであろう．例を挙げると (1) 2つの言語でプログラマの予備知識や経験に差があること，(2) プログラムすべき問題についてのプログラマの予備知識や経験に差があること，(3) プログラマの固有の熟練度に差があること，(4) 問題が一方の言語に組込まれれた能力にぴったり合っており，他方の言語にはそうでない．これらは2つの言語を比較することのむずかしさのほんの一部に過ぎない．初めの3つの

† 厳密にはソースとオブジェクトはコンパイラの入力と出力となるプログラムを指す．オブジェクト言語は機械語である必要はない．

要素は，テストで観察される人々に関係がある．4番目は，たとえば，次のような状況で説明される．逆行列を求める問題でCOBOLとAPLを比較すると仮定しよう．APLではこの問題を，言語の持つ1つの演算子で処理できるが，COBOLでは(ほとんどの他の言語と同様に)かなりずむずかしいプログラムを必要とする．結局，言語性能の客観的評価は現在，計算機科学の最前線に位置しているといえる．

少し乱暴ではあるがトランスレーション性能は2つの数によって有益な特徴付けがなされる．その1つはコンパイラの要求する主記憶空間であり，他の1つはコンパイル速度であり，プログラムのある特定の集合の平均値としての1分当りコンパイルされるステートメント数で測定される．普通，速度だけが引用されるが，測定されたステートメントが動作負荷として"典型的"なものと信ずるかどうかはわれわれに任されている．コンパイラが要求する主記憶の量はコンパイラの必要とする記憶装置の総量である必要はない．コンパイラの要求する記憶装置の多くを主記憶とせず，コンパイラ，そのテーブル，ソース・プログラムなどの一部だけを必要に応じて主記憶装置にもってくるといった，コンパイラの"折り重ね"が広く行なわれている．この手法を用いて，非常に複雑な言語に対し，優れた診断能力を持つコンパイラを小さな計算機にも提供できる．コンパイラの設計に際して，少なくとも仮想記憶でない環境下では，コンパイラが使用可能な主記憶の量はその計算機に大きな影響を与える基本的なパラメータである．コンパイラ以外の注意深く書かれた多くのプログラムと同様に，使用可能な主記憶が多くなればなるほど，より速いコンパイラ・プログラムが設計できる．

最近実際に行なわれていることから少し例を引いて上述の議論を説明しよう．IBMシステム/360オペレーティング・システム設計では，広く用いられているFORTRAN Ⅳ言語に少なくとも3つのコンパイラ，FORTRAN F，GおよびHが作られた．これらの文字は(コンパイラの必要とする)主記憶サイズが64k，128k，そして256kバイトであることを示す．他の例はPL/1(F)コンパイラで，広く使われている言語で多分最も複雑なものであるPL/1ソースプログラムを翻訳するのに64kバイトの主記憶装置しか要求しない．このコンパイラは何度も折り重ねられ，何度もディスクをアクセスしながら，ソース・プログラムを20回以上読むことが必要となることも多い．

ある主記憶サイズとコンパイラ性能の関連は仮想記憶(第10，11章を参照せよ)の出現で，なくならないまでも，少なくなりそうである．仮想記憶システムのコンパイラ設計は現在発展しつつある技術で，集中的に探究されている．

翻訳速度はこの節の始めに掲げられた性能の大きな分類の1つに過ぎない．これに最も近いのはオブジェクト・コード性能である．これら2つの項目は，今から述べるように，ある程度釣合いがとられる．直感的に，他の条件はすべて同じとして，コンパイラが効率的なオブジェクト・コードを生成しようとすればするほど，何度もソース・プログラムを解析しなければならず，より長い翻訳過程を経ると考えられる．逆に，結果としてオブジェクトは比較的遅い速度となるがコンパイラに高速なコンパイレーションをさせることも可能である．当然，与えられたコンパイラの設計がどちらをとるかはコンパイラがどう使われるかに依る．商工業のユーザでは(1度デバグ，コンパイルされた)プログラムは異なったデータで幾度も，幾度も実行されるのが典型的であるので，多くの計算機製造会社はそこでは主としてオブジェクト・プログラムを走らせると見ている．このため製造会社から供給される多くのコンパイラは速いオブジェクト・コードを生成するもので，コンパイル速度はかなり遅いのも驚くべきことではない．

大学や研究所では，その計算機活動全体のかなりの量が，開発済みのプログラムの生産/実行よりもむしろ，プログラム開発であることが多い．この典型的なものは学生や研究者が新しい言語を学習したり，新しい手法を探究したりする時である†．いくつかの大学では率先して高速コンパイルするコンパイラを作製した．よく知られている最近の2つの例は FORTRAN コンパイラの WATFOR(カナダのワーテルロ大学で作られた)とコーネル大学で設計，コード化された PL/1 コンパイラ，PL/C である．両者とも非常に速いコンパイルと優れた診断・援助を特色とし，しかも驚くことに，トランスレータの性能を減じていない(同じ動作負荷，装置での生産向きのコンパイラに比較して)．しかし高速コンパイラにはいくつかの制限がある．たとえば，PL/C は PL/1 言語の部分集合でしかなく，ファイル・アクセシング，**ベースつき記憶域**(based storage)や一般性(キーワードを識別子，または“名前”として用いることのできる能力)のようないくつかの特力な械能を除いている．PL/C と WATFOR は共に“コア常駐”のコンパイラである．コンパイラのすべては，コンパイル過程の間，しかもオブジェクト・プログラムの実行中も主記憶装置にある(コア常駐)．PL/C はコンパイルとオブジェクト実行時の間，約 100k バイトの

† 工業的な使用と大学/研究の使用の差はここでは議論のきっかけとしようとしたものである．各機関の計算機使用の種類の概括を意図したものでない．たとえば，多くの大学の計算センタの活動では大部分のジョブは学生に対する型のものであるが，他の大部分のシステム利用は行政や他の型の生産的な処理である．

主記憶装置を必要とする．一般にコア常駐のコンパイラは全部の翻訳のジョブを一時に行うことを条件とする．それゆえ，別にコンパイルされた他のプログラムを処理中のプログラムにリンクするための再配置可能なコードや中間形式をとる必要がない．この理由から，“リンク-エディット”と“ロード・ジョブ-ステップ”(OS/360 の用語)は除かれている．それらに対応するもの(翻訳プロセスの一部として遂行される)は非常に簡単で高速である．最後にコア常駐コンパイラは1ジョブずつ順次ジョブをとり込み，コンパイルする各ジョブ毎にコンパイラを主記憶装置に運ぶためのオーバヘッドを除くことで，コンパイル速度をさらに高めことができる．コア常駐コンパイラの主な欠点は，オブジェクト実行時のスペース効率の悪さである．その理由は，コンパイラがその時有効な仕事していなくて，それである限られた大きさの主記憶装置に対して，置かれるオブジェクト・プログラムの大きさに制限を与えるからである．

コンパイラ性能の3番目の分類項はコンパイラが生成するオブジェクト・プログラムが要求する“**領域と時間**(space time)”である．**コンパイラ最適化**という言葉は一般にこの種の能力を達成するための手法を記述するのに使われる．この課題は集中的に研究，開発が行なわれた．1972年の時点で利用できる手法についていくつかのまとまった報告がだされた†．今から少しばかりの紙面で，いくつかの手法を概説し，それらの性能改善の能力の程度についての感触を与える．

オブジェクト性能の達成可能な値の見込みを得るために，**図6-6.1**を考察しよう．図はFORTRANで書かれた科学計算型の4つの問題のプログラムのオブジェクト実行時間を示している．プログラムはIBM OS/360 FORTRAN Gコンパイラと3つの最適化方式によるFORTRAN Hコンパイラによって翻訳された．また比較のためモデル50の実行時間も示されている．Gコンパ

	オブジェクト実行時間(秒) モデル 65 IBM FORTRAN コンパイラ				モデル 50	
問題	G	H 0	H 1	H 2	G	H 2
100,000 個の加算	1.02	1.07	0.94	0.20	3.94	0.80
式の演算	0.07	0.08	0.07	0.05	0.29	0.18
行列積($N = 128$)	0.35	0.48	0.38	0.13	1.26	0.47
高速フーリエ変換($N = 512$)	0.32	0.27	0.31	1.25	1.10	0.35

図 6-6.1　4つの科学計算における IBM FORTRAN コンパイラ，モデル 50，モデル 65 の比較

† 参考文献 GR, RU に概説と，より広い参考書名が載っている．

イラは飛び抜けてとはいえないがかなり優れたコードを生成する．一方，FORTRAN H 2 は (1960年代中頃作られたにもかかわらず) 今でもなお最適化技術の現状を反映している．実際の数字よりもっと重要なことには，図 **6-6.1** には注目されなければならないある傾向や，その反対の傾向が示めされている．まず第1に，実行時間比は，たとえば FORTRAN G と FORTRAN H 2 の間では，問題に依存し，H 2 で翻訳されたプログラムは 5.1 倍速いものから，たった 1.13 倍のものまである．もちろん，そのようなコンパイラのオブジェクト効率の比較は同じマシン上で行なわれなければならない．しかし，良いコンパイラを用いることで，(ある問題に対しては) より速いマシンを劣ったコンパイラで用いたのに匹敵する改善がなされることに注目しよう．たとえば，同じコンパイラを用いるとモデル 65 はモデル 50 の約 3.6 倍速いが，行列積プログラムに対してモデル 50 の H 2 コンパイラを用いて翻訳されたコードはモデル 65 の G コンパイラで生成されたプログロムより単に約 1.3 倍遅いだけである．しかしながら，高速フーリエ変換プログラムについては，H 2 で生成されるコードは G のよりほんのわずかだけ速いだけはので，3.6 のマシン/速度比は G を用いたモデル 65 に比べて H 2 でのモデル 50 でほんのわずか (3.1 に) 変っている．それで要約すると，図 6-6.1 の少ないサンプルから少なくとも，5 から 1 のオーダの速度の改善は現代のコンパイラ最適化手法を用い，ある種のプログラムに対して達成し得る．しかし，あるプログラムはほんのわずかしか改善されない．また，もちろんソース・プログラムとデータに依っては 5 以上の倍数の可能性を除外してはいない．

言語-コンパイラの効率の 3 つの分類を手短かに議論してきたが，ここでわれわれは最適化についてもっと綿密に調べることにする．各分類項間の関係もまた議論されよう．まず手始めに，かなりはっきりしていることを述べる．すなわち，最適化を行なうコンパイラによって生成されたコードは，翻訳手法とマシンのリソースを熟知した非常に有能なプログラマによって作られる最良のコードより速くはなりえない．これは極端な場合を言っているのであって，多くのプログラマはそれらの条件を満たさないのでそれは (実際には) 当てはまらないことを記しておく．本当の問題は，現実の環境で理論的に正しいコードを期限に追われながらつくろうとしている多くのプログラマより，コンパイラはうまく最適化できるであろうかということである．その答はイエスという響きがかえってくる．

最適化手法の 1 つはあるステートメント中の冗長な計算を除くことである．例として，次の PL/1 または FORTRAN ステートメントを考えよう．

$$R = A*X*X*X*X + B*X*X*X + D$$

$X*X*X$ が2度現われることを知って2つの積を簡略化できる．そこで，上の式は次のように書き直される．

$$T = X*X*X$$
$$R = A*X*T + B*T + D$$

式の一部が同一であることを知ることが最適化に役立つ他の例としては（見つけるのがかなりむずかしいが）数個のステートメントの中での共通な式がある．たとえば，2次方程式の2根を求めるよく知られたステートメントを考えよう．

$$R1 = -B + (B**2 - 4*A*C)**0.5$$
$$R2 = -B - (B**2 - 4*A*C)**0.5$$

これらは次のように書き直され，実行時間で約2倍節約されるであろう．

$$T = (B**2 - 4*A*C)**0.5$$
$$R1 = -B + T$$
$$R2 = -B - T$$

おそらく最も重要な最適化手法はプログラム・ループに関するものである．最も単純な技法はループ中のステートメントでループの実行に対して**不変**（invariant），すなわち，その値が（最初のパスを除いて）ループを回っても変らないものを見つけることである．ループから不変のステートメントを取り除くことは，プログラムの論理を壊すことなく1回の実行がN回の実行に置き代るため，明らかに実行速度を改善できる．

ジャミング（jamming）と呼ばれるループの凝縮の手法は，2つ以上のループを1つにまとめることである．プログラム B はプログラム A のジャミングの結果を示している．両プログラムは同じジョブを行なう．すなわち，ベクトル X と Y のすべての要素を0にする．

プログラムA

```
   DO 10 I = 1,100
   X(I) = 0.0
10 CONTINUE
   DO 20 I = 1,100
   Y(I) = 0.0
20 CONTINUE
```

プログラムB

```
   DO 10 I = 1,100
   X(I) = 0.0
   Y(I) = 0.0
10 CONTINUE
```

もちろん，もしそもそもプログラム B がソース・プログラムだったら，同じオブジェクト・コードが生成されるだろうが，この時はコンパイラによる改善はないだろう．要するに，現代のコンパイラ手法は“無頓着に”(だらだらと)コード化したソース・プログラムにはかなりの速度の改善をもたらしうるが，注意深く書かれたコードには(性能の観点から)はるかに少ない改善しかもたらし得ない．上述のジャミング(ループの融合)手法はループをまとめようとし，それで冗長な“指標の更新”と“テスト分岐”を除いている．2つのループがまとめられるためには，次のことが満たされれば充分である．すなわち，(1) 片方のループが実行されるなら，もう一方のループも実行される．(2) 片方のループでの計算は，もう一方の計算に影響されない．(3) 両方のループは同じ回数だけ実行される．ジャミングはオブジェクトの実行時間を減らすだけでなく，必要とされる記憶容量をも減らす．

その他のコンパイラ最適化手法はループを**展開**することである．単純なループは“直線的に”，すなわち，そこで実行されるすべての命令で，書けることは少し考えればわかる．たとえば，次の2つのFORTRANプログラムを考えよう．

プログラムA　ループ	**プログラムB　直線**
DO 10 I = 1,4	Z(1) = X(1) + Y(1)
Z(I) = X(I) + Y(I)	Z(2) = X(2) + Y(2)
10 CONTINUE	Z(3) = X(3) + Y(3)
	Z(4) = X(4) + Y(4)

プログラム A では，2番目のステートメントだけが本質的である．DO文は初期化命令(レジスタ・ロード)と(各ループを回るごとに)指標の更新，テスト，および分岐に翻訳される．プログラム B は多くのステートメントを書くことを代償に，ループ制御命令の実行を避けている．展開の手法は直線的手法の速さを取り戻そうとしている(そして，若干の記憶容量の損失を受ける)が，それでもループはその利点を多く保持している．たとえば，プログラム D はプログラム C を2倍に展開したソース・プログラムに対応する．

プログラムC	**プログラムD**
DO 10 I = 1,100	DO 10 I = 1, 100, 2
Z(I) = X(I) + Y(I)	I1 = I + 1
10 CONTINUE	Z(I) = X(I) + Y(I)
	Z(I1) = X(I1) + Y(I1)
	10 CONTINUE

2つづつに展開することでの時間的な有利さは高々2倍である．多くの場合，しかしながら，その有利さははるかに少ない．(上の例では，浮動小数点加算命令が支配的で，展開されたものもされていないのも両方で同じ回数だけ実行される．)展開するむずかしさの1つは指標の限界値が上の例のように定数(100のように)でなく変数であるときに起こる．その場合でもなお展開は用いられるが，ループの正しい実行を保証するため(ループ外に)付加的なステートメントを挿入しなければならない．

さてオブジェクト実行時間の最適化のいくつかの手法をまとめよう．

1. ループからの不変なステートメント除去，不変なステートメントとはループの中でそのステートメントの結果の変数がループ実行で変わらないものである．
2. 単一のステートメント中の，またはステートメント間の冗長な式の除去．
3. 数個のループを1つのループにジャミング，すなわち統合する．それによる，冗長なループ制御の実行を減す．
4. ループ制御の実行を省くためにループを展開する．
5. **のぞき穴**(peephole)最適化．コンパイラの生成したオブジェクト・コードを走査してより効率のよいシーケンス，またはマイクロコードのシーケンスにおきかえられるシーケンスを探すことによって行なう．
6. 代替の“**文法的解釈**(parsing)”．これは並行処理やパイプライン方式のマシンで特に有益でありうる．たとえば

$$Z = A + B + C + D$$

を次の2つに解釈することを考えよう．

$$P1: \ Z = (((A + B) + C) + D)$$
$$P2: \ Z = ((A + B) + (C + D))$$

ここで $P1$ は直列的に実行されなければならないが，$P2$ を用いると2つの加算器を持つシステムで並列的に $A + B$ と $C + D$ は実行されうる．

最適化を行なう現代のコンパイラの巧徴さが時々驚異的なパフォーマンスの結果をもたらしうることをついでに述べておこう．たとえば，ある設備ではCPU速度の比較に用いられる“ベンチマーク”プログラムの1つはループ中にいくつかのステートメントを含むような式の計算である．そのステートメントはいろいろな型の算術演算を既知数含み，ループ制御変数を含まないように作られているループを，たとえば100,000回実行させることで，計算機速度は

ストップ・ウォッチで測ることができる．この方法は粗雑だが価値ある結果が得られ，その手順はユーザの制御下にあり，テストされるシステムの製造者の制御下にはない．しかし，ここで最適化を行なうコンパイラによってそのようなプログラムが翻訳されたときどんなことが起こるか考えて見よう．時間を測定するステートメントはすべて不変なもので，ループの外に出され，ただの1度しか実行されない！ そこで観測されるシステムの速度は，コンパイラのおかげで，しかもプロセッサの速度にほとんど関係なく，非現実的なほど速いものとなるだろう！

コンパイラ設計者はその仕事の手助けに役立てうるいろいろな技法を持っていて，そのいくつかは上で簡単に議論された．質的に，最適化の代償の1つはより遅いコンパイル速度である．さらに，使えるすべての手法の中で，ある手法が他のものよりずっと有益であることは疑わしく思われよう．どれが最良かという問題はコンパイルされるプログラムに依存するのも当然である．また一方，ある手法はほとんどの場合に非常に実のあるものであろう．これらの考察から共通の考えが浮かぶ，すなわち，プログラムの実行時の性質に関するデータを集め，書かれた静的なプログラムとそれらを関係づける．これを行なう手段はプログラムのよく動く範囲を見分けるのに役立つ．もし多くのプログラムに共通のパターンがあれば，それらを見つけることでコンパイラ設計者は，コンパイラを改良するために欠くことのできない情報を得るだろう．おのおののユーザも同様な手段を用いることはその人のプログラムの最も動きの大きな部分に配慮の焦点を合わす手助けとなる．そしてこれらの部分のパフォーマンスを改善することに集中できる．

スタンフォード大学の Dan Ingalls と Don Knuth はこの種の結果について報告している．彼等は**エグゼキューション-タイム・プロフィル・モニタ** (Execution-Time Profile Monitor) と呼ばれるある種のソフトウェア道具を開発した．このプログラムは解析される FORTRAN ソース・プログラムを入力として受取る．これを処理していろいろな種類のカウンタ(たとえば，あるステートメントが実行されるごとの，またあるクラスのステートメントが実行されるごとのカウンタ) を更新するステートメントを付加した新しい FORTRAN ソース・プログラムを生成する．実行後，カウンタの情報は頻度表あるいは**ヒストグラム** (histogram) としてプリントされる．Ingalls はあるプログラムが35個のサブルーチンを持ち，その中の1つだけで実行時間の67パーセントを占める場合(非常に典型的であるとは疑わしいが)を引用している．これでそのプログラムに注意の焦点を合せ，さらに少しばかりの検査で彼等は内

最適化レベル	レベル4に対する相対実行時間	インライン・サブルーチンを除いた場合のレベル4に対する相対実行時間
0	4.6	3.7
1	3.4	2.7
2	1.9	1.5
3	1.3	1.1
4	1.0	0.8

最適化レベルの定義

最適化レベル ＝1　局所的最適化：共通な部分表現を除く．レジスタ内容をステートメントを越えて保持する．
＝2　大局的最適化：不変な表現を除く，ロード-ストア動作の最適化など．
＝3　レベル2と同様，但し，IBM S/360 のマシンに依存した最適化を行なう．(BXLE，LA，2重インデックスなどを用いる)
＝4　知られているコンパイラ最適化をすべて用いる．

サンプルのプログラム数：　"20以上"
プログラムのソース言語：　FORTRAN Ⅳ
手法：　プログラムの非常に頻度の高い"内部ループ"を見つけるためにソフトウェア・トレース（エグゼキューション-タイム・プロフィル）を用いる．それらのループを各最適化レベルで手作業でシミュレートし，計時する．

図 6-6.2　実行時動作に対するスタンフォード大学の研究の概要 (D. Ingalls [15] による報告)

部ループの不変なステートメントを見つけた．それをループの外にただ動かしただけで，彼等は40分のランを15分にまで下げた！　**図6-6.2**には Ingalls によって引用された他の研究の結果を示す．今回の特徴解析は数個のプログラムの内部ループを見分けるのに用いられた．そして**5**つのコンパイラ最適化技法の手によるシミュレーションが行なわれた．表に見られるように，最善のコンパイラ方策は最も簡単なものより4.6倍の改善がなされている．これは当然，**図6-6.1**で述べたIBMのFORTRAN GとH2コンパイラ間の最も好都合のときの比率に近い．図6-6.1では改善度はプログラムに依存したことが想起される．

エグゼキューション-プロフィル・アナライザ(execution-profile analyzer；略して **EPA**）はプログラムのパフォーマンスに対するプログラマの認識を改善するのに優れた道具である．プログラミングの仕事は複雑なので，書かれたプログラムと実行されるプログラムとの間に非常に差があることと相まって，プログラム性能についてわれわれの新たな直観的洞察を働かせねばならないことになる．EPAは学生，本職のプログラマ，そして管理者にとってさえ，プログラム速度の改善に最も役立つ仕事に焦点を合せる手助けとなりうる．マイクロプログラムとして実現するのに適当なコードを識別するのに非常な手助けにもな

るであろう．コンパイラはユーザの選択でEPAを与えるように設計されうる．

言語，翻訳，オブジェクト性能の3つの性能分類を導入したが，後の2つについては多くのことがわかっているので議論の大部分はそれらについて行なった．この簡単な概観を終えるにあたって，他の2つの要素に留意しなければならない．第一に，最適化技法は基本的にそれが何をしているかに関するソース・プログラムから抜き出された情報に依存する．よりレベルの高い言語であればあるほど，ユーザのプログラムがユーザが**何を**行ないたいかにもっと集中し，ユーザはそれが**どのように**処理されるかをコンパイラに任せるので，情報の抽出は容易になりうることは，当然想像される．たとえば，APLの強力な行列演算子を用いて，ユーザは単一のステートメントで行列の積や逆行列の演算を指定でき，それで簡単なしるし(単一の演算子記号)でコンパイラ設計者にこれらの演算を最適化する最大の機会を与える．対照的に，たとえばFORTRANやPL/1では行列の積や逆行列演算のプログラムは多くのループからなりそれとわかるのは難しい(特にあるプログラマのコードの書き方を)．配列向き高級言語からの翻訳の場合はまた並列処理マシンやマイクロコード化されたマシンに対するコンパイルにも非常に有益である．

最後に，この章は性能の考察のみに関したものであったことを言っておかなければならない．他の要素，たとえば論理的明快さ，信頼性，ドキュメンテーションのような要素もまたプログラムがうまく働くために非常に重要である．

演　習　問　題

1. 磁気コア・メモリのアクセス時間とサイクル時間の違いについて簡単に説明せよ．

2. メモリ性能の尺度としての帯域の意味を論ぜよ．またなぜその値がアクセス時間やサイクル時間より優れた値なのかを説明せよ．

3. サイクル時間1マイクロ秒，モジュール当りの幅16ビットの16モジュール磁気コア記憶サブシステムを考えよ．
   (a) 1つのモジュールの帯域はいくらか？
   (b) システムの最大帯域はいくらか？
   (c) システムの最小帯域はいくらか？
   (d) 実効帯域を見積れ．

4. 幅32ビット，サイクル時間800ナノ秒のコア記憶装置モジュールを考

えよ．400ナノ秒の実効サイクル時間を得るにはインタリーブされるモジュールがいくつ必要であるか？

5. あるジョブ・ストリームに対するある計算機の動きは次のような命令実行の統計で示される．

命令タイプ	時間/命令，マイクロ秒	相対頻度
ロード/ストア	1	0.40
加算/減算/比較	1	0.30
分岐	1	0.25
乗算	8	0.04
除算	12	0.01

(a) 平均命令実行時間はいくらか？　その逆数を平均命令実行速度とする．

(b) もし除算時間が4倍改善されたら，何パーセントの速度の改善となるか？

(c) もし加算時間が4倍改善されたら，何パーセントの速度の改善となるか？

(d) (c)で加算時間が改善されるとその結果，一般に乗除算も改善される．加算時間が4倍改善されると乗除算が2倍改善されると仮定する．この条件下で加算時間が4倍改善されると何パーセントの速度の改善となるか？

6. カーネル法に比して命令ミックス法の優れている点，劣っている点を述べよ．

7. マイクロコード化されるべき情報を識別するための実験的手法について記せ．

8. エミュレーションの本質と重要性について論ぜよ．

9. あるレポート生成プログラムは長さが80バイトである問題レコードを扱い，その処理時間としてレコード当り2ミリ秒を要求する．レコードは10ミリ秒のアクセス(起動-停止時間)の2つの磁気テープ装置から読まれ，そのフロー率は100kバイト/秒である．処理が済むと，レコードは再びテープ上に格納される．

(a) ブロック化係数を20とすれば，どれだけの量のデータ記憶装置

が要求されるか？　また完全にシーケンシャルなシステムではレコード当りの平均処理時間はいくらか？

（b）　入力，出力と計算がオーバラップされるシステムでは(a)はどのような値となるか？

10. 問9の構成に対して最適なブロック化係数，レコード当りの総記憶容量と処理時間を求めよ．

11. なぜコア常駐型コンパイラは折り重ね型のものより高速であるのか，少なくとも2つの理由を示せ．

12. コンパイラにより生成されたコードはアセンブリ言語のプログラマにより書かれた最良のコードより速くないのに，なぜアセンブリ言語でプログラミングすることが広く行なわれないのか？

13. 次の最適化の方策について論ぜよ．

（a）　ジャミング　　　（b）　展開

（c）　不変値(式)

14. エクゼキューション・プロフィル・アナライザ(EPA)とは何か？　EPAを作成する2つの方法：コンパイラにそれを埋め込む方法とソース・コードを変換する方法の2つを比較・検討せよ．

15. プログラミング言語の持つ特徴がコンパイラの最適なコード生成にどのように手助けとなるか論ぜよ．

参考文献

1. [AL] Allen, F.E. and J. Cocke: A Catalogue of Optimizing Transformations, in "Design and Optimization of Compilers," R. Rustin, ed., Prentice-Hall, Englewood Cliffs, N.J., 1972.
2. [AN] Anacker, W. and C.P. Wang: Performance Evaluation of Computing Systems with Memory Hierarchies, *IEE Trans. Comp.*, vol. EC-16, no. 6, pp. 764-773, December, 1967.
3. [AP] Apple, C.T.: The Program Monitor, A Device for Program Performance Measurement, *Proc. ACM 20th National Conf.*, August, 1965.
4. [BA] Baldwin, F.R., W.B. Gibson, and C.B. Poland: A Multiprocessing Approach to a Large Computer System, *IBM Systems Journal*, vol. 1., no. 1, pp. 64-76, September, 1962.
5. [BE] Bell, G. and A. Newell: "Computer Structures," McGraw-Hill Book Company, New York, 1971.
6. [CA] Calingaert, P.: System Performance Evaluation: Survey and Appraisal, *Comm. ACM,* vol. 10, no. 1, January, 1967.
7. [ES] Estrin, G., D. Hopkins, B. Coggan, and S.D. Crocker: Snuper Computer: a Computer in Instrumentation Automation, *AFIPS Conf. Proc.*, vol. 30, pp. 645-656, 1967 *SJCC.*

8. [FU] Fuchi, K., H. Tanaka, Y. Manago, and T. Yuba: "A Program Simulator by Partial Interpretation," *Second ACM Symposium on Operating System Principles*, Princeton University, pp. 97-104, October 20-22, 1969.
9. [FU1] Fuller, S.H.: Minimal-Total-Processing-Time Drum and Disk Scheduling Disciplines, *Comm. ACM*, vol. 17, no. 7, July 1974.
10. [GI] Gibson, J.C.: The Gibson Mix, *IBM Technical Report TROO.2043*, June, 1970.
11. [GR] Gries, D.: "Compiler Construction for Digital Computers," John Wiley and Sons, New York, 1971.
12. [HE1] Hellerman, H.: "Digital Computer System Principles (2nd edition)," McGraw-Hill Book Company, New York, 1973.
13. [HE2] Hellerman, H. and H.J. Smith, Jr.: Thruput Analysis of Some Idealized Input, Output, and Compute Overlap Configurations, *ACM Computing Surveys*, vol. 2, no. 2, June, 1970.
14. [HU] Husson, S.S.: "Microprogramming Principles and Practice," Prentice-Hall, Inc., Englewood Cliffs, N.J., 1970.
15. [IN] Ingalls, D.: The Execution-Time Profile as a Programming Tool, in "Design and Optimization of Compilers," R. Rustin, ed., Prentice-Hall, Englewood Cliffs, N.J., 1972.
16. [KN1] Knight, K.E.: Changes in Computer Performance, *Datamation*, vol. 12, no. 9, pp. 40-54, September, 1966.
17. [KN2] Knight, K.E.: Evolving Computer Performance 1963-1967, *Datamation*, vol. 14, no. 6, pp. 31-35, January, 1968.
18. [OP] Opler, A.: Measurement of Software Characteristics, *Datamation*, vol. 10, no. 7, pp. 27-30, July, 1964.
19. [RA] Raichelson, E. and G. Collins: A Method for Comparing the Internal Operating Speeds of Computers, *Comm. ACM*, vol. 7, no. 5, pp. 309-310, May, 1964.
20. [RO] Rosin, R.F.: Contemporary Concepts of Microprogramming and Emulation, *Computing Surveys*, vol. 1, no. 4, 1969.
21. [RU] Rustin, R. (ed.): "Design and Optimization of Compilers," Prentice-Hall, Englewood Cliffs, N.J., 1972. (Consists of seven papers and bibliography.)
22. [TU] Tucker, S.G.: Microprogram Control for System/360, *IBM Systems Journal*, vol. 6, no. 4, pp. 222-241, 1967.
23. [WH] White, P.: Relative Effects of Central Processor and Input-Output Speeds Upon Throughput on the Large Computer, *Comm. ACM*, vol. 7, no. 12, pp. 711-714, December, 1964.
24. [WI] Wilkes, M.V.: Microprogramming, *Proc. Eastern Joint Computer Conference*, December, 1958.
25. [WN] Winder, R.O.: A Data Base for Computer Performance Evaluation, *Computer*, vol. 6, no. 3, pp. 25-29, March, 1973.
26. [WO] Wood, D.C., and E.H. Forman: Thruput Measurement Using a Synthetic Job Stream, *AFIPS Conf. Proc.*, vol 39, pp. 51-56, 1971.

第7章

オペレーティング・システム
——その発展と基礎——

ほとんどの汎用計算機は一般的な目的を達成するように設計されたいくつかのシステム・プログラム群の下に動作する．その目的とは次のものを含む：

1. 物理的なI/O装置を直接名前で呼びプログラミングしたり，記憶装置やI/O装置の割当てを行なうなどの，かなり詳細で長々とした仕事からプログラマを引き離すこと．
2. ジョブの開始と終了の際に要求される人間であるシステム操作員の労力を減らすこと．
3. たとえば，最も高い優先度を持つジョブを最初にスケジュールするというようなある規準に従って，ユーザに良いサービスができるようにジョブをスケジュールすること．
4. CPU，主記憶装置，I/Oチャネル，I/O装置などのハードウェア資源をできるだけ有効に使うこと．
5. 複数のユーザやジョブにCPU，主記憶装置，プログラム，データベースなどのシステム資源の並列的な共有を可能とすること

これらの目的(およびその他の目的)は程度に差があるが今日のオペレーティング・システムの多くで達成されている．15年から20年前のシステムに比較して，現在のオペレーティング・システムは小さなコンピュータのものでさえかなり凝っているように思える．しかしながら，まだ改善の余地がある．多くの主記憶装置管理技法は主記憶装置を浪費し，**細分化**(fragmentation)の問題を引き起す．またジョブ-スケジューリングの詳細なパラメータが一般的に人間のスケジューラや操作員によって余りに多く与えられなければならず，オペレーティング・システムによって自動的に決定されえない．情報保護と機密管理の機構もまた現在のところかなり初歩的なものである．

この章ではオペレーティング・システムの機能的な発展の概略を述べる．われわれはオペレーティング・システムを持たない計算機の操作を記述することから始める．最後に考察するクラスのシステムは一般的な多重プログラミングの能力をもったものである．パーフォマンスはその目的としたことからのみ記述しうるものなので，何を目的としたかということからこのマルチプログラミングの発展をたどることにしよう．ある目的，またはその集合を最適化するという観点からオペレーティング・システムの設計を考えることが最善であろう．その目的はしばしば述べられているが，オペレーティング・システムの理論と実際は不幸にも，そのような文章が実際に設計パラメータを決めるのに充分なほど正確なものであるようにはまだなっていない．

いろいろなオペレーティング・システムで見られた幾つかの重要な機能について議論しよう．可能な限り，この本で導入したスケジューリングの原則と性能尺度の適用性を示そう．オペレーティング・システムに関する全般的な参考文献は [DO, FR 1, HU 3, SE 1, ST 1] にある．

7-1　オペレーティング・システムのない計算機

コンパイラ(アセンブラを含めて)の他にはソフトウェアのないシステムを考えて見よう．この環境は歴史的に興味深いが，しかし今日でも小さなシステムでは時々遭遇する．またオペレーティング・システムの特徴の本質を理解するための概念上の出発点としても興味深い．

プログラムはユーザとシステムとの間の橋渡しであるので，プログラムのいくつかの特質を手短かに振り返ることから始める．計算機の CPU は論理的な主記憶装置から直接命令を取出しデータにアクセスするだけである．したがって実行中のプログラムは少なくとも一部分が主記憶装置になければならない．この実行中のプログラムの記述は機械語でなされていなければならない．しかし，プログラムは機械語で書かれず，むしろある記号的な原始言語で書かれている．どのような原始言語で書かれた原始プログラムでも論理的なデータや命令を指定する記号的な**名前空間**(name space)を参照すると考えられうる．コンパイル時にコンパイラは名前を番地に翻訳する，すなわち，名前空間はアドレス空間に変換される．プログラムのアドレス空間には機械命令と実行中に参照される可能性のあるすべてのデータ項目と命令のための場所がある．コンパイラによって生成されるプログラムのアドレス空間は**オブジェクト・モジュール**(object module)(これも IBM 用語である）と呼ばれる．プログラムのオブジェクト・モジュールがさらに翻訳され，そして最後に実行のため主記憶装

置に移されるという過程がどのように行なわれるかを，“オペレーティング・システムなし”から一般的な多重プログラミングのものまでの機能的な発展の各段階ごとに議論しよう．

オペレーティング・システムがない場合には，ユーザのジョブは一時に1つ実行されるので，各ジョブは実行中，計算機システムのすべてを独占する．**オープン-ショップ**(open-shop)操作と呼ばれる機構では，システムは各ユーザに彼自身のジョブを走らせるため物理的に引き渡される．それに比して，**クローズドーショップ**(closed-shop)操作ではすべてのジョブは**オペレータ** (operator)と呼ばれる1人の人によって走らされる．その人は**装置**に明記された操作手引きに従うよう強いられており，システムに対して会計上，組織上の責任をもっている．いずれにしても，計算機システム全体はスケジュールされる1つの資源であるから，一度次のジョブが選ばれると，ジョブは終るまでシステムを完全に占有すると考えられる．

クローズドーショップ操作を仮定すると，オペレータに求められるスケージューリングに関する決定は，ただ次に何のジョブが扱われるかを選択することである．ジョブを始めるために，オペレータは穿孔カードや磁気テープなどの入力媒体から主記憶装置へオブジェクト・モジュールの**ローディング**を開始し，それからジョブのプログラム実行が実際に始まる．この手順はマシンの特別なボタン押す(そしてダイヤルをセットする)ことで物理的に起動される IPL(initial program load)で開始される．これは少量の情報を(たとえば)カード読取機から主記憶装置の0番地からに移す作りつけのマシン・シーケンスを起動する．それからプログラム・シーケンスは0番地から開始する．このプログラムは典型的な“ブートストラップ”であり，始めにカードや補助記憶装置(ディスク)上に入れられた写しからブートストラップの残りを主記憶装置に読み込む．自分自身を読み込み終った時点で，ローダはその主たる仕事であるわれわれのジョブのオブジェクト・モジュールの主記憶装置への読み込みにとりかかる．**アブソリュート・ローディング**(absolute loading) と呼ばれる最も簡単な型のシステムでは，オブジェクト・モジュールのアドレスの修飾はされない．これはコンパイラによって生成されるアドレスはまた物理的なマシン・アドレスであるということである．それでプログラムのアドレス空間はコンパイル時の主記憶装置に**縛られている**と言う．そのようなプログラムは，後で議論される**再配置可能な**(relocatable)プログラムに対比して，ときどき**アブソリュート**なプログラムと呼ばれる．オブジェクト・モジュールの読込みに引き続いて，ローダはそれに分岐し，そしてジョブ自身が実行を開始する．

プログラム中で直接名付けられており主記憶装置中にあると見なされる命令やデータに加えて，多くのプログラムは実行中にI/O装置から得たり，送出したりしたデータも参照する．オペレーティング・システムなしでは，I/O装置の名前は普通コンパイル時に物理的なI/O装置に結びつけられる．装置の割当てを変更するためには，まづソース・プログラムの変更を行ない，それから新たにコンパイルすることを必要とする．記号的な名前と物理的な名前を論理的に結びつけることに関するこれらの機能やこれに似たような機能は，オペレーティング・システムのサービスの欠くことのできない役割と思われる．次の節でこの基本的な問題を調べることにしよう．

資源の共有という観点からは，オペレーティング・システムのないマシンは共有をしないので全く不充分なものである．すなわち，各ジョブはそれが走るときすべてのマシンの資源を独占する．それで各ジョブはそれが使う可能性のある限りできるだけ多く資源を指定する強い誘因がある．しかしながら，たとえあるジョブがその実行中にすべての資源を使うことがあっても，その実行時間の多くでは主記憶装置やI/O装置の一部分のような資源は遊んでいることを留意しよう．

より大きな主記憶装置を持ってより速い動作をさせることが経済的に可能となる点にまで技術は進んだので，個々のプログラムの多くはほとんどの資源を大部分の時間，生産的に用いることができない．それゆえ，計算機のリソースをいくつかの**“実行可能な”**プログラムの間で，すなわち，同時に主記憶装置中にあるものの間で並列的に共有しようとする誘因は増大する．これは多重プログラム・オペレーティング・システムの理論的根拠であり，本章の後の方と次の章で議論される．

7-2　単一プログラム・オペレーティング・システム［HU 1 参照］

たとえ計算機が一度にただひとつのジョブしか処理しなくても，その動作や有効性を高めることができる多くの機能がある．制御プログラムは次のような機能を提供できる：

1. 再プログラミングや再コンパイルなしで2つ以上のオブジェクト・プログラムを結合して新たな実行モジュールを作る能力．
2. 磁気テープ上のファイルのレコードをプリンタに出力したりファイルのレコードをソーティングしたりするような共通のタスクを行なうユーティリティ・プログラム
3. 最小限のオペレータの介入でジョブからジョブへの容易な遷移

4. ユーザ・プログラムやシステム・プログラムをたとえば直接アクセス記憶装置のようなシステムのライブラリに入れておく能力
5. I/O 装置をコンパイル中でなく実行直前に割当てる能力．このように後で結びつけるのでコンパイル・ランを節約でき，実際の装置の可用性にこたえるものである．
6. 各ジョブが各システム資源を用いた時間の会計．

これらのような目的から直接に**単一プログラム・オペレーティング・システム**(monoprogrammed operating system)のクラスが導びかれる．**単一プログラム**の意味はどのようなサービスを受けようともジョブはひとつづつしか実行されず，あるジョブが終了するまで次のジョブが始められないということである．

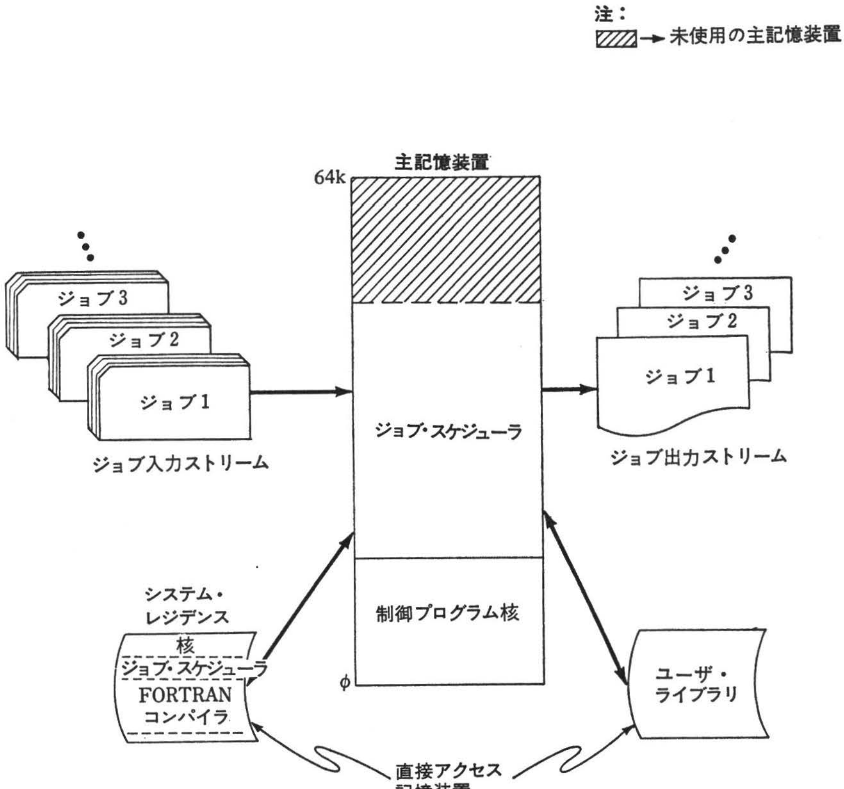


図 7-2.1 単一プログラム・オペレーティング・システム

図7-2.1で示される状況を考えよう．穿孔カード・デック（ジョブ入力ストリーム），ジョブを処理するために使われるシステム資源(図では主記憶装置や直接アクセス装置として示されている)やジョブの出力プリントとしてジョブの集合が描かれている．上述の機能がどのように実現されるかを示すため，まずシステム内のジョブの流れを追うことにしよう．

ジョブ入力ストリームは処理されるジョブの集合で，おのおののジョブはその名前で他と区別され識別される．ジョブとは他のジョブと論理的に独立な仕事の単位を識別するために使われる用語である．ジョブは1つ以上の**ジョブ・ステップ**から成ると考えられる．ジョブ・ステップとは簡単にはジョブを論理的に細分化したものである．たとえば，まずプログラムの翻訳(コンパイル)をし，それからその実行を要求するジョブを考えよう．そこでは，第1のジョブ・ステップはコンパイルであり，第2のジョブ・ステップはコンパイルされたプログラムの実行(しばしば**GO ステップ**と呼ばれる)である．

図7-2.2のように，ジョブ入力ストリームは穿孔カードに入っている．最初のカードは**ジョブ制御**型のもので名前でそのジョブを識別し，また多分課金請求用の番号のような他の情報も与えるものである．(同一ではないが)同じようなカードを用いて各ジョブ・ステップで使われるプログラムを少なくとも名前で示して，そのジョブ・ステップを識別しなければならない．そのカードの後に各ジョブ・ステップで用いられる必要なすべてのI/O装置や前もって入れられたデータの集合を指定するための1枚以上のカードがある．この一連の指定はオペレーティング・システムで用いられる**ジョブ制御言語**（job-control language；略して**JCL**）の規定に従って明記される．ジョブ制御言語を用いた

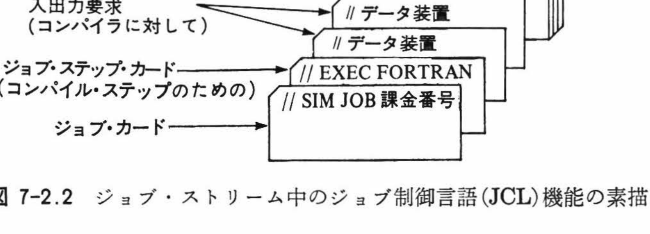


図 7-2.2　ジョブ・ストリーム中のジョブ制御言語(JCL)機能の素描

文(簡単に JCL と呼ばれる)はジョブを識別し，システム・リソースに対する要求の指定を行なうものと考えられる．穿孔カードの入力データを要求するジョブ・ステップはそのジョブ・ステップの JCL につづくジョブ・ストリーム中にデータを持っているであろう．図 7-2.2 では JOB SIM と呼ぶジョブの一連のジョブ制御カードがどのようになっているかを示した．ジョブ入力ストリーム中の各ジョブはこれと同じようなジョブ制御カードを持つであろう(IBM OS/360 ジョブ制御カードの議論を行なっている 8-1 節を参照のこと)．

上述のジョブ入力ストリーム(すなわち，ジョブの待ち行列)は物理的にシステムのカード読取機に存在する．それでジョブの選択は少なくとも読取機のジョブ・デックの間では到着順(**FCFS**; first-come-first-service)である．(もちろんオペレータはカード読取機に置くジョブの順番を定める種々の規則を使うことができる.)

さて典型的な単一プログラムのオペレーティング・システムのジョブ・スケジューリングの機能に移ろう．この機能は図 7-2.1 に示された**ジョブ・スケジューラ**と呼ぶシステム・プログラムによって行なわれる．ジョブ・スケジューラ・プログラムはシステム常駐装置(普通にはディスクのような直接アクセス装置)に入っているが，常に主記憶装置中にある必要はない．典型的には高価な主記憶装置の領域を節約するために，スケジューラの**核**だけが主記憶装置中に置かれ，そして各ジョブ・ステップの終りに次のステップをスケジュールするために，その核はスケジューラのコピーをシステム常駐装置から主記憶装置に持ってくる．このシステムでは FCFS 法則が用いられ，スケジューラは単にカード読取機から次のジョブを読むだけなので，ジョブの選択は簡単なことである．ジョブ・スケジューラはジョブの一部分，すなわち一度に 1 つのジョブ・ステップをシステムに読込む．ジョブ・ステップの JCL(図 7-2.2参照)を読む間に，ジョブ・スケジューラはその JCL 文を解釈し，そのジョブ・ステップで使われるプログラムを識別し，そのジョブ・ステップが用いるすべての I/O 装置を識別する．そして(もし特定の装置がそのジョブ・ステップの JCL 文中で識別されていなければ)そのジョブ・ステップに I/O 装置を割付けたり，そのジョブ・ステップが要求するプログラムをロードしたりするサービスを，ジョブ・スケジューラは**制御プログラム核**(control program nucleus)に要求する．プログラムのローディングは制御プログラムで行なわれる．これはロードされるべきプログラムのオブジェクト・モジュール(またはアドレス空間)が直接アクセス装置のユーザ・ライブラリに(図 7-2.1 のように)入って

いると仮定する．オブジェクト・モジュールはジョブ・ストリーム中の穿孔カードにもまた存在し得る．どちらの場合でも，ローディングの後では，そのプログラムはジョブ・スケジューラをオーバレイしており，割当てられたI/O装置に結びつけられていよう．もし何らかのオペレータの動作(たとえば，磁気テープを装填するとか)が必要とされるのならば，制御プログラムは適当なメッセージを用いてオペレータに知らせる．それからそのジョブ・ステップの実行が開始できる．**図7-2.3**にはこの時点の主記憶装置の構成が示されている．そのジョブ・ステップが実行されている間は，他のジョブやジョブ・ステップはシステムを使用できない．実行中ではどんなシステムのメッセージもジョブ・ステップの出力もシステム・プリンタに打ち出される．ジョブ・ステップが終ると，制御プログラムに知らされる．そして制御プログラムが主記憶装置へ(システム常駐装置から)ジョブ・スケジューラのコピーをロードし，そのジョブ・ステップをオーバレイし，またジョブ・スケジューラは(今述べたのと同じ手順に従い)次のジョブ，またはジョブ・ステップをスケジュールする．

このようなシステムはオペレーティング・システムのないものに比べて多くの利点を持つ．

1.　ユーザ・プログラムはユーザ・ライブラリの中に蓄えられ，JCL文によ

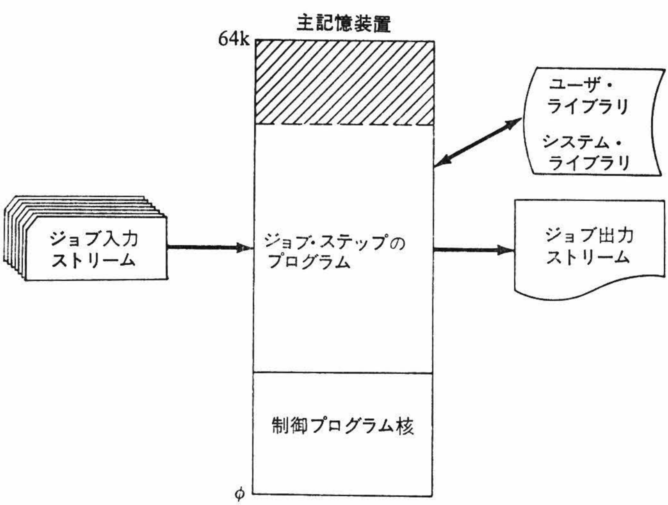


図 7-2.3　ジョブ・ステップ実行時の主記憶装置

ってライブラリから呼ばれる．これで穿孔カード・デックに多くのユーザ・オブジェクト・モジュールを入れる必要を除いている．

2. ジョブ制御言語がディスク上のライブラリから前もって入れられているシステム・プログラムを呼ぶ，というような多くの詳細な機能を行なうので，オペレータに，したがって，遅くて誤りを犯しやすい人間の操作に余り頼らない．
3. ここではジョブがストリームを形成し，おのおののジョブは自分自身の組立ての大部分を指定する JCL 文を持っているので，よどみなくジョブからジョブ，またはジョブ・ステップからジョブ・ステップへの移行でき余りオペレータに依存しない．
4. I/O 装置はコンパイル時にではなく，むしろプログラム・ロード時にプログラムと結合される．

制御プログラムはジョブやジョブ・ステップで用いられた CPU 時間，壁時計の時間(一日の時刻)，主記憶装置，I/O 装置などを監視(記録)することも可能である．この情報はジョブの課金，請求に使うことができる．これらの機能に必要な CPU ハードウェアにはリアル・タイム・クロックやインターバル・タイマ(7-4 節参照)が含まれる．

ジョブ制御言語は言語であるから，よく使われる一続きの JCL 文 (JCL サブルーチン)を直接アクセス装置の JCL 手続ライブラリに入れることが可能であり，一続きの文はジョブの JCL がシステムに読込まれたとき 1 つの文で呼ばれる．よく使われるユーティリティ・プログラムはまたシステムやユーザ・ライブラリに入れられ，JCL で実行のため呼ばれる．JCL はまたプログラム名と前もって入れられたデータとの結合を後で行なうことを許す．

これまで単一プログラム・オペレーティング・システムを，システムを通り抜けて行くジョブを追うことにより調べた．その中で制御プログラム核やジョブ・スケジューラなどの必要とされる多くのソフトウェア・リソースで遂行される機能を述べた．システムを眺める他の方法はそのソフトウェア・リソースを通して眺める方法である．

図 7-2.1 に示めされるように**システム常駐装置**は直接アクセス装置に設けられる．このシステム常駐パックには制御プログラム核，ジョブ・スケジューラ，システム・ソフトウェア・リソース(コンパイラ，ソート，ユーティリティなど)，ブートストラップ・ローダ・プログラムなどのコピーが入っている．それらはすべて実行形式(オブジェクト・モジュール)である．

さてオペレーティング・システムがどのように“開始”されるか考えよう．

(ただしシステム常駐パックが存在していること，そして，それは前もって作られたものであるか他のシステムにあるものと仮定する.)マシンの電源が投入されたとき，人間であるオペレータの最初の仕事はオペレーティング・システムの "IPL" である．この時点ではマシンのオペレーティング・システムは動作していないので，事実上オペレーティング・システムのないときと同様のIPL の処理(前の節で述べられている)が行なわれる．一般的には操作員はコンソールのボタンを押したりダイヤルを回してシステム常駐装置を指定し，それから IPL ボタンを押すことによって，その装置から主記憶装置への情報の転送を起動し，次いで，読み込まれた情報に分岐させる．これは単にブートストラップ・ローダであり，順々にオペレーティング・システムの核を読込む．普通には次にジョブ記録のため使われる時刻，日付などの情報を入れるようオペレータに要求する．これに続いて，ローディング・プログラムはその核の一部分に分岐する．われわれは，それを単に制御プログムと呼ぶことにするが，それは必要に応じていろいろなオペレーティング・システム・プログラムにリンクする．われわれは核という用語を大雑把に制御プログラムと同義語として用いることにする．

IPL 手続きの後で，核はジョブ・スケジューラのコピーを主記憶装置にロードする．そしてジョブ・スケジューリングが前述のように始められる．コンパイラやユーティリティのような要求されるソフトウェア・リソースはいずれもJCL によって指定され，必要なときにシステム常駐パックからロードされる．同様な方法で，システムによってはユーザ・ライブラリの中に応用プログラムも入れることができ，JCL 文で要求できる．これにより共通に使われるプログラムを各ユーザがシステムに与える必要性を減らす．

プログラム処理の観点からは，単一プログラム・オペレーティング・システムは次のように見える．プログラムは初めあるソース言語でコーディングされる．つぎにそのプログラムはシステム常駐パックに入れられているコンパイラを用いてコンパイルされる．プログラマはコンパイラを適当な JCL 文で呼ぶ．コンパイルした出力はオブジェクト・モジュール，すなわちユーザのアドレス空間である．ここで各アドレス空間は 0 から始まると仮定する．制御プログラム核は主記憶装置中に 0 番地から入っているので，オブジェクト・モジュールは実行のためにロードされる前にさらに変換する必要がある．**リンケージ・エディタ** (linkage editor) と呼ばれるシステム・プログラムはしばしばアドレス変換のために用いられる．それは二つの型の事柄を行なう．

1.　2つ以上のオブジェクト・モジュール，すなわちすでにコンパイルされ

たプログラムを結合し，**ロード・モジュール**(load module)と呼ばれる単一アドレス空間を生成する．

2. オブジェクト・モジュールのある種の情報に対し必要な変更を施すので，その結果であるロード・モジュールはリンク・エディト時に指定される主記憶装置の任意の箇所に置くことができる．

コンパイラがある取決めに従うオブジェクト・モジュールを生成するように設計されているときのみ，リンケージ・エディトが可能である．これらの取決めは外部名，すなわちある手続き(プログラム)で使われる名前で，その手続きの外部のプログラムやデータを参照するもののすべてを特定の形で識別することに集約される．それらの名は普通，**外部記号辞書**(external symbol dictionary)と呼ばれる表に置かれ，リンケージ・エディタは種々のオブジェクト・モジュールを一つのアドレス空間に一緒に結合するため，この表によってこれらの名前を識別することができる．アドレス定数や命令の値が主記憶装置中のモジュールの位置に依存する(そのため変更されなくてはならない)ので，それらを識別するために同じような指定が必要である．これらの値を変える過程はしばしば**再配置**，この場合には**静的再配置**(static relocation)と呼ばれる．

そこで論理的プログラム・アドレスと物理的主記憶装置領域との結合はリンク・エディット時に行なわれる．単一プログラム・オペレーティング・システムでは，すべてのプログラムは同じ場所から順にロードされるとするのが普通合理的であるが，この場合にはアドレス結合には(静的再配置を行なう)リンケージ・エディティングで充分である．多重プログラミング・システムでは，リンク・エディット時に主記憶装置とアドレスを最終的に結合することは，(後で議論されるように) 主記憶装置割当てにかなり柔軟性を欠く結果となる．しかしながら，やはりこの場合でもリンク・エディティングは結合の重要な中間形式である．

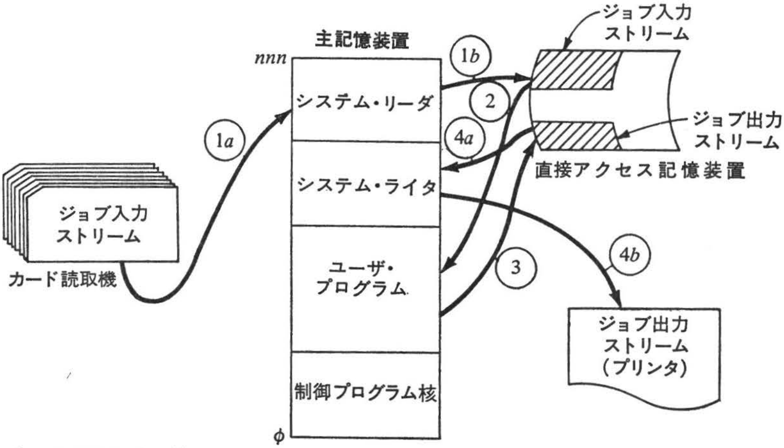
用意された多くの新しい機能に対して，単一プログラム・オペレーティング・システムはハードウェアあるいはソフトウェアのいづれのリソースも余り多くは，要求しない．ハードウェア・リソースには主記憶装置の一部と直接アクセス記憶装置が含まれる．制御プログラム核はシステム動作中主記憶装置にある．ジップ・スケジューラはジョブやジョブ・ステップの間で主記憶装置を使う一時的なシステム・リソースである．直接アクセス装置はシステム常駐用，システム・ライブラリ用，およびユーザ・ライブラリ用に用いられる．ソフトウェア・リソースはこの節で述べられた特定のプログラム，すなわち，制御プログラム核，ジョブ・スケジューラ，IPL プログラム，リンケージ・エ

ディタ，コンパイラ，および(ソート・プログラムのような)特殊目的のユーティリティ・プログラムを含む．これらのリソースの構造と相互作用は単一プログラムおよび多重プログラムのオペレーティング・システムの基礎である．

最も簡単な単一プログラム・オペレーティング・システムは逐次的にその仕事を行なう．ジョブの待ち行列はカード読取機にできる．ジョブあるいはジョブ・ステップがカード読取機から選ばれると，それはシステム全体を制御する．すなわち，まずカード入力を直接カード読取機から取り，次に主記憶装置中で処理を行ない，印字出力を直接プリンタに転送する．6-3節に述べられたように一つのプログラム中で入力，計算，出力はオーバラップできるが，それは特定のプログラムだけに可能であり，プログラマに特別の努力を要求する．もっと一般的には，単一プログラムはカード読取機，プリンタ，CPUをしばしば遊ばせることになる．もし良いスループット(ある時間中で行なわれる仕事)が主たるシステムの目的ならば，これらの遊び時間を減らす生産的な技法がこの目的に合致するだろう．これはシステム入力，ジョブ処理，システム出力を**オーバラップ**させるシステムで可能である．この型のオーバラップは**スプーリング**(**SPOOLing**; Simultaneous Peripheral Operation On Line)と呼ばれる．スプーリング技法は次節で述べられる．

7-3 多重プログラミング：スプーリングのみのシステム

スプーリングの主な目的はシステムのスループット改善である．これはシステム入力，ジョブ処理，システム出力をオーバラップさせることで達せられる．システム入力は前節で述べられたジョブ入力ストリームのことである．システム出力はここではカード穿孔，印字出力をさし，高速プリンタで印字されるJCLリスト，システム・メッセージ，ユーザのジョブ出力を含む．ユーザ・ジョブは一度に1つ実行すると仮定する．スプーリングには2種類の特別なシステム・プログラムが主記憶装置に常駐している必要がある(図7-3.1参照)．ひとつは**システム・リーダ**(または単に**リーダ**)と呼ばれる．その機能はジョブ入力ストリームをカード読取機から直接アクセス記憶装置に移すことである．(1*a* と 1*b* 段階)．第2のプログラムは**システム・ライタ**(または単に**ライタ**)と呼ばれる．その機能は直接アクセス記憶装置からのシステム出力を書く(印字する)ことである(4*a* と 4*b* 段階)．スプーリングは実際にカード装置やプリンタ(そして時々テープも)などの速度の遅い装置を直接ユーザが制御することをなくしているが，これは実行上であって，論理的な構造からではない．すなわち，各ソース・プログラムはスプーリングなしで行なうのと同じ様に論理的に



機　能： 1a-システム・リーダ
カード→主記憶装置
1b-システム・リーダ
主記憶装置→直接アクセス記憶装置
2- ユーザ・プログラム
直接アクセス記憶装置→主記憶装置
3- ユーザ・プログラム
主記憶装置→直接アクセス記憶装置
4a-システム・ライタ
直接アクセス記憶装置→主記憶装置
4b-システム・ライタ
主記憶装置→プリンタ

図 7-3.1 システム・リーダとライタ機能

“カードを読み”そして“出力を印字する”.

スプーリング・システムで達成されるオーバラップ（そして多重プログラミング）は**入出力チャネル**によって可能となる．チャネルはハードウェア装置で機能的にはI/O装置から主記憶装置に入力を，主記憶装置からI/O装置に出力を送るように設計された特殊目的の小型計算機と類似のものである．チャネル動作を開始するために普通はCPU命令が必要であるが，一旦始まると，速度の遅い転送やそれらのバッファ動作などはCPU命令の実行と無関係にチャネルによって行なわれる．CPU動作の大部分とチャネル動作が独立していることはそれらが同時に，すなわち時間的にオーバラップできることを意味する．

そこでスプーリングの重要な機能は直接アクセス記憶装置を中間段階として，それによりシステム入力，出力をバッファすることである．カード読取機とプリンタを同じ時刻には異なったジョブに割当てることで，できるだけ同時に動作させるように意図したリーダ，およびライタと言うシステム・プログラムによりバッファリングは制御される．これらのシステム・プログラムはユー

ザ・プログラムと主記憶装置を共有するので，スプーリングは多重プログラミングの原始的な形である．

スプーリングは装置と直接アクセス記憶装置との間の(主記憶装置を介して)システム I/O の自動転送を特徴とする．スプーリングは(たとえばファイルの更新のような)主記憶装置と直接アクセス記憶装置との間の転送を要求するユーザ・プログラムの動作とは論理的に独立している．

スプーリングはジョブ入力ストリームを直接アクセス記憶装置に移すので，ジョブ待ち行列は実質的には直接アクセス記憶装置上にある．ジョブをスケジュールするため，ジョブ・スケジューラはジョブ待ち行列から次の候補として最良のものを捜す．“最良”とはスケジューリング・アルゴリズムによって定義される．直接アクセス装置はカード読取機と異なり順次型の装置ではない．それゆえジョブ・スケジューラは FCFS スケジュール規則だけに限られない．たとえば**静的優先度順**(**HSFS**；highest-static-priority-first-served)スケジュール規則が使えるであろう(実際にしばしば使われている)．

典型的なジョブ・スケジュール処理が図 7-3.2 に示めされている．ジョブ・スケジューラは次に実行されるジョブを選ぶ．最初のジョブ・ステップで使われるプログラムはジョブ・スケジューラのコピーをオーバレイして主記憶装置にロードされる．実行中，穿孔カード入力は実際には直接アクセス記憶装置から読まれる．その直接アクセス記憶装置にはリーダによってカード・イメージが入れられている．(これは図 7-3.1 の 2-段階に対応する)．すべての印字出力も直接アクセス記憶装置の別の場所に入れられる(3-段階)．

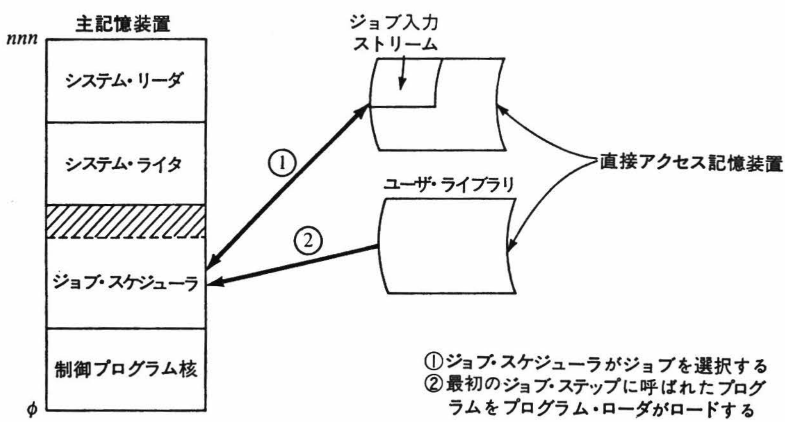


図 7-3.2　スプーリング・システムにおけるジョブ・スケジューリング

ある回数の間，ジョブ出力は印字されるより高速に直接アクセス記憶装置に蓄積される．余分の情報はプリンタがそれを受付けることができるようになるまで記憶装置中にとどまる．当然ながらもし非常に長期間蓄積速度が印字速度より速いとその蓄積をさせる処理は一時停止させられなくてはならない．ライタはジョブからの出力の印字を制御する．ライタが次の印字のジョブを選択するのに(他の規則も使えるが)FCFS 規則を用いるとすれば，カード読取機，プリンタ，CPU が同時に動作するという意味で，ライタはユーザ・プログラムとシステム・リーダと共に同時に動作できる．一旦ジョブがライタに選択されたら，ライタはレコード(一行の印字イメージ)を出力ジョブ・ストリームから高速プリンタに送る(図 7-3.1 の 4*a*-と 4*b*-段階).

これまで入力，出力，CPU 処理の別々のハードウェア装置が，低速の I/O 動作を含んでいるかなりの時間にわたって，どのように同時動作できるかを見てきた．しかし装置を制御するチャネルでさえ CPU の世話(すなわち，それらを起動すること)を必要とすることが理解された．そこでリーダ，ライタおよびユーザ・プログラムの要求をスケジュールする技法が必要である．この種の CPU のスケジューリングはよく**ディスパッチング** (dispatching) と呼ばれる．スプーリング・システムにおけるディスパッチングは制御プログラム核によって行なわれる．使用される技法は一般に**横取使用**(preemption)のある静的優先度の規則である．CPU の時間開始は横取と割込みの作用で制御される．ここしばらくの間, 割込みは 2 種類の事象，すなわち (1) I/O の要求 (2) I/O 動作終了時チャネルによるもの，で起こると仮定する．

リーダとライタは元来 I/O 処理プログラムである．それらが制御する装置は低速であるから，装置を絶え間なく作動させておくことがスループットを良くするのに必要である．装置の新たな動作を開始するため短い一瞬の間 CPU を必要とすることが時々あるだけなので，リーダやライタはユーザ・プログラムより高い優先度を持つ．優先度は一般に次の順で与えられる；(1) システム・リーダ (最も高い) (2) システム・ライタ (3) ユーザ・プログラム (最も低い)．そこでリーダは常に最初に CPU を使える可能性がある．もしリーダがなすべき仕事を持っていたら，自身が I/O 要求を出し，割込みを起こすまで CPU の制御を保持する．そして特権命令の *START I/O* で制御プログラム核はリーダのためにチャネルを起動する．これでカード読込み処理が始まりそれから核はライタ (実行準備ができていると仮定する) に CPU 制御を渡たす．ライタがプリンタに対する I/O を始めた後で，CPU の制御は問題プログラム(実行準備ができていると仮定する)に渡される．さもなければ CPU は待

ち状態に入る．リーダのI/O動作が終わった時(第2のタイプの割込み)，核を通してCPU制御はリーダに戻る．

この型のディスパッテングは**事象駆動型** (event driven) である．ディスパッテングを行なう事象は割込みである．リーダ・プログラムがCPUを必要とする時はいつでも(たとえ他のプログラムでCPUが使われていても)，リーダ・プログラムはCPUを取るので横取りであり，静的優先権方式を用いる．リーダやライタはI/Oにより制限される傾向にあるので，普通ユーザ・プログラムを実行する機会は充分ある．スプーリングだけのシステムは初歩的であるが非常に重要な多重プログラミングの例である．しかし，それらのシステムは基本的な欠点を持っている．すなわち，一時にひとつのCPU実行可能な**ユーザ**・プログラムしか主記憶装置にないことである．そのプログラムがI/Oを待たなければならない点まで達したら，(比較的まれなCPUに対するスプーリング要求を除いては)CPUは遊んでいる．このCPUの遊びは主記憶装置に同時に複数個の他のユーザ・ジョブを入れておくことで避けられる．この方法はより一般的な多重プログラミング・システムとなる．計算機技術の発展に伴ないCPU速度や主記憶装置の増大は計算機技術の発展に役だってきたが，これはまたマルチプログラミングを適切なものとしている．

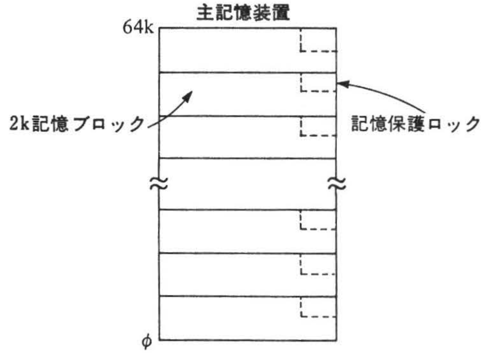
7-4　多重プログラミング：ハードウェアの特徴

スプーリング・システムでは空間および時間はシステム・リーダ，システム・ライタおよびユーザ・ジョブに共有される．共有される空間は主記憶装置とシステムI/Oバッファ用のディスク空間であり，時間はCPU時間である．より一般的な型の多重プログラミング・システムでは主記憶装置中に同時に存在する2つ以上のユーザ・ジョブの間ですべてのシステムの資源，すなわちCPU，主記憶装置，I/Oチャネル，I/O装置，およびソフトウェア資源を共有できる．システム・リーダとシステム・ライタを“ジョブ”と考えれば，スプーリングも多重プログラミングの一種である．しかし，一般的な多重プログラミング・システムでは(リーダやライタのような)システム・プログラムに加えて2つ以上の**ユーザ・ジョブ**がシステム資源を同時に共有することを許す．われわれはまず，多重プログラミング・システムで要求される特別なハードウェアについて述べよう．はっきりした例はIBM 360と370から得られるが，これらの特徴は(少し変形されて)他の多くのシステムで見られる．多重プログラミング・システムを実現するのに欠くことのできないと，最近，実際面で考えられている4つのハードウェア機能は

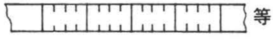
1. 記憶保護(主記憶装置)
2. 特権命令
3. 割込方式
4. インターバル・タイマ，または実時間クロック

である．最初の項目，**記憶保護**はあるユーザ・プログラムや制御プログラムが占有している記憶領域に他のユーザ・プログラムが情報を書き込めないようにしている．この“書き込み保護”に加えて，あるシステムでは“読み出し保護”も行なっている．すなわち，ユーザ・ジョブがそれに割当てられた主記憶装置の領域を超えて読み出すことを禁じている．記憶保護には記憶が参照されるごとに認められたアクセスかを調べる必要がある．主記憶装置の高速動作のためにはこのチェックはハードウェアでなされなければならない．さもなければシステムの速度は耐えられない程遅くなるであろう．

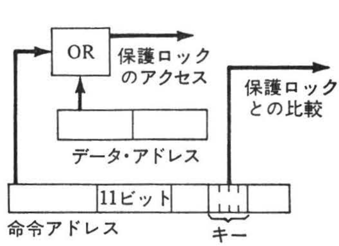
記憶保護を行なうために用いられる1つの技法はキー-ロック機構である．**図 7-4.1** の (*a*) は IBM システム/360 で用いられている機構を図に表わしたものである．議論を明確にするため，2k 毎に分割され，おのおのにロックをもつ 64k バイトの主記憶装置を考えよう．図の(*b*)は各ロックの構造を示し，4 ビットからなることがわかる．4 ビットのロックはすべて特別のロック・ストア用記憶装置に入れられる．4 ビットのロックなので $2^4 = 16$ 個のロックが可能であり，それゆえ主記憶装置中で同時に16個までの異なる記憶領域を保護できる．



(*a*) 2kバイト記憶ブロックごとの保護ロック



(*b*) 保護ロックの配列



(*c*) キーを示すプログラム・ステータス・ワード

図 7-4.1　記憶保護；キー・ロック機構

ユーザ・プログラムが主記憶装置に割当てられる前に，制御プログラムによって記憶装置に“キー値”が与えられ，それからそのユーザプログラムに割当てられた記憶装置のロックのために未使用のキー値の一つが与えられる．ある与えられたプログラムを起動したり再開始したりする前に，オペレーティング・システムは**プログラム-ステイタス・ワード**(program status word；略して **PSW**)と呼ばれるCPUレジスタの4ビットの保護キーのフィールドにプログラムのキー値を置く．プログラムが主記憶装置にデータを格納しようとする時は必ず，PSW内のキーは参照すべき記憶ブロックにあるロック・ビットと比較される．もしロックとキー値が一致しなければ，ストア操作は行なわれず，**保護割込**(protection interrupt)が起きて制御プログラムを呼び出す．そして制御プログラムはたとえばプログラムを終了させ，ユーザにエラー・メッセージを与えるような動作をするであろう．0のPSWキーは特別な場合で，この値が置かれるとプログラムは主記憶装置のすべてを参照し変更できる．多重プログラミング・システムでは制御プログラムは0のキーが割当てられる．これにより，制御プログラムは主記憶装置の任意の位置を参照し変更できる．ユーザ・プログラムはそれに割当てられた主記憶装置領域の中しか変更できない．また制御プログラムだけがロックを含んでいる記憶の内容とPSWキー・レジスタを変更する特権を持つ．

多数のユーザのプログラムがシステム資源の共有を許す多重プログラミング・システムでは，その制御プログラムは主なシステム資源の割当てと使用を制御しなければならない．たとえば，制御プログラムだけがロックやキーを置く命令を実行できなければならない．すべてのI/O割当ても同様に制御プログラムによってのみ管理されなければならない．IBMのシステム/360と370では，これは**特権命令**(privileged instruction)によって行なわれる．CPUのPSWレジスタには**モニタ・ビット**と呼ばれる1ビットがあり，実行中のCPUプログラムが特権命令を実行してもよいことを示す．CPUが制御プログラムを実行しているときにはこのビットはONである．ほとんどの割込み処理の初めの方でこのビットがONにされ，そして制御プログラムがCPUの制御を他のプログラムに渡す前に制御プログラムによってOFFにされる．制御プログラムの実行中ではないことを示すモニタ・ビットがOFFならば，特権命令を実行しようとする試みはすべて失敗し，制御プログラムに割込みが生じる．

ある記憶保護ブロックにキーを置く“SET STORAGE KEY”は特権命令の一例である．またCPUがチャネル操作を起動するための“START I/O”も一例である．それで制御プログラムのみがすべてのI/O動作を制御でき，

複数のユーザからの装置(たとえば，データ・チャネル)に対する要求に答えることができる．もしチャネルがある1人のユーザのプログラムにそのジョブの継続期間割当てられていたら，そのチャネルに接続されているすべてのI/O装置はその一人のユーザに専有される．要するに，制御プログラムだけがシステム資源をユーザの仕事に割当てることができるように特権命令で保証しているのである．

制御プログラムは，互いに参照される関係で一緒に結びつけられたシステム・プログラムの集合であり，割込み信号で実行の開始を“刺激される”．**割込み機構**はこの信号が起こされる機構である．IBM システム/360 とシステム/370 の割込み作用は次の優先順位を持つ五つのレベルを持つ：

1. **マシン割込み**　ハードウェア誤動作によって生ずる．マシンのチェックが起きたら，一般にオペレーティング・システムは動作を継続しない．
2. **プログラム割込み**　プログラムの誤りによるもの，たとえばゼロで割ろうとしたとき．
3. **スーパバイザ・コール (SVC) 割込み**　ユーザ・プログラムによるシステムのコントロール・プログラムのサービス要求によるもの．たとえばI/O 動作の開始．
4. **外部割込み**　特別なシステム装置から生じるもの．一例としては，インタバル・タイマがプログラムで定められる時間間隔の終りにゼロとなり外部割込み要求が起こる．
5. **I/O 割込み**　I/O 動作の終了，あるいはカード・リーダのようなI/O装置が動作可となったとき起こる．

制御プログラムが事象駆動型で静的優先順位のディスパッチングを用いるとき，I/O 割込みと SVC 割込みがまず CPU ディスパッチング (あるいはスケジューリング) に含まれる．これは IBM の OS/MVT のような多量プログラミング・システムでの一般的なディスパッチング手法である．タイムシェアリング・システムのように時間単位が CPU ディスパッチングに用いられるときには，**インタバル・タイマ**あるいは**実時間クロック**が必要となる (第9章参照)．一般的な多重プログラミング・システムの記述を進める中で今述べたハードウェアの使用について言及しよう．

7-5 多重プログラミング：時間と空間の共有 [CA, DE, FR 参照]

どのような記憶装置でも，空間のみ，時間のみ，空間と時間の両方の共有が

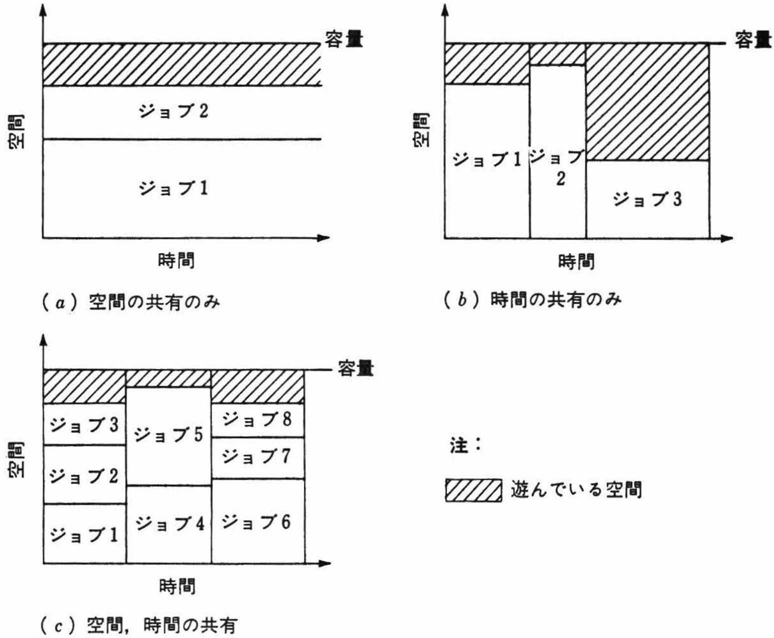


図 7-5.1　コンピュータの主記憶装置：時間空間の共有

可能である．主記憶装置は多重プログラミング・システムにおいて最も重要な記憶装置資源である．**図 7-5.1** の(a)は主記憶装置の**空間的共有のみ**を示している．すなわち，同じ記憶域にはあらゆる時間同じ情報が入れられている．そのような状況は連続的なオンライン・プロセスの制御システムでは珍しくはない．主記憶は時間的に共有されていないが，CPU 時間は常駐タスクによって共有されていることに注意しなければならない．

主記憶装置の**時間的共有のみ**の場合を図 7-5.1 の(b)に示す．これはオペレーティング・システムのないコンピュータで主記憶装置が一時に1人のユーザ，あるいは1つのプログラムに使われる状況である．図の(c)は一般的な多重プログラミングで実現できることを示す．複数のユーザ・ジョブが主記憶装置を同時に占有し，そして古いユーザ・ジョブが終了したとき新しいユーザ・ジョブが主記憶装置中にロードされる．このことからジョブ j の主記憶装置使用の尺度としてその**時間-空間積**が考えられる．すなわち，ジョブ j に対し確保した空間のバイト数 S_j と空間が確保されている時間 T_j を掛けたもの

$$P_j = T_j S_j$$

プログラムのアドレス空間やコンパイラによって生成される目的ジュールの概念についてはすでに述べた．オペレーティング・システムのないときアドレス空間は一般にコンパイル時に主記憶装置のロケーションに結びつけられる．単一プログラム・オペレーティング・システムではプログラムのアドレス空間はその後のリンケジ・エディット時に主記憶装置のロケーションに結びつける手法がよく行なわれる．スプーリング・システムでも同じ手法が用いられるものと仮定する．3つの例のすべては，要するにプログラムのローディングあるいは実行の前に主記憶装置に結びつけられたプログラムを用いている．これは単一プログラミングでは満足できるものだが，多重プログラミング・システムでは主記憶装置の使用効率が悪くなる．その理由は実行時に主記憶装置を実際にどう占有するかをコンパイル時に予測できないからである．たとえば，4つのジョブがジョブ待ち行列にあり，各ジョブが主記憶装置の領域に重って割当てられているなら，一時にただ一つのジョブしか実行のため主記憶装置にロードされない．数個の待っているジョブをロードするのに充分な一続きの空間があるときですらそうである．この結果主記憶装置が遊ぶ．

多重プログラミングでもっと有効な手法として，プログラムをロードする時点（この時にはコントロール・プログラムは使用可能な主記憶装置の状態を知っている）まで，プログラムのアドレス空間を主記憶装置の連続領域に割当てないものである．この手法は**静的**あるいは**ロード時再配置**と呼ばれる．この考えを**図 7-5.2** に示す．同じプログラムのアドレス空間がプログラムがランされるごとに主記憶装置の異なった領域に割当てられることもある．

静的再配置では，ローダ・プログラムはプログラムのアドレス空間でのアドレスを主記憶装置の物理的なロケーションに変換しなければならない．この機能を持っているので，われわれはそのローダを（図 7-5.2 に示されるように）再

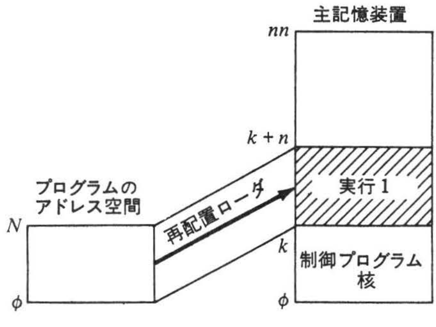
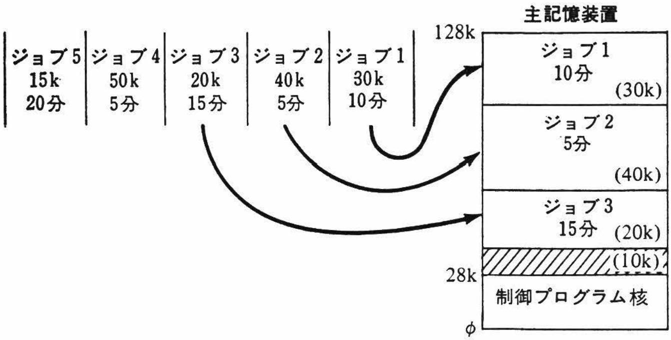


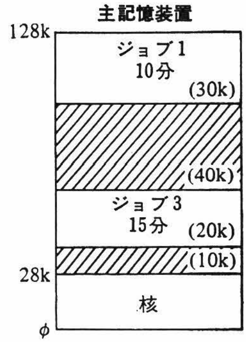
図 7-5.2　静的（ロード時）再配置

配置ローダと呼ぶ．アドレスはロード時に**のみ**変換，あるいは再配置されるので，一旦開始されたプログラムはその実行中ずっとその主記憶装置のロケーションに結びつけられている．たとえプログラムの実行が割込まれ，そのプログラムが主記憶装置から直接アクセス記憶装置に移されても，実行を再開するにはそのプログラムを主記憶装置の同じ領域に戻さなければならない．

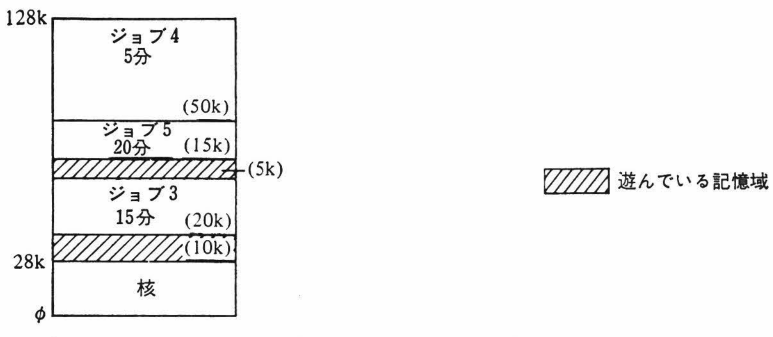
図 7-5.3 では短いジョブ・ストリーム例を用いて主記憶装置管理のための静的再配置技法を説明している．この例では，各ジョブは 1 つのプログラムを呼ぶ 1 つのジョブ・ステップを持っている．プログラムに必要な時間，空間はジョブ名の下に書かれている．主記憶装置はジョブに対し上から下に割当てられ，ジョブは FCFS 規則でスケジュールされていると仮定する．主記憶装置の大きさは 128 k バイトで，最初の 28 k はコントロール・プログラムの核に用



(a) 最初のジョブがスケジュールされたところ (FCFS 規則)



(b) 5分後 (新たなジョブはスケジュールされない)



(c) 10分後(二つのジョブが新たにスケジュールされる)

図 7-5.3　静的再配置による記憶装置管理

いられているとする．初めの3つのジョブは直ちにスケジュールされる([a])．しかしJOB2が終ったとき，主記憶装置は**細分化**されている([b])．すなわち，使われていない空間が多くあるが，それらが連続しておらず，細片に分かれている．JOB1とJOB3はおのおの主記憶装置のロケーションに結びつけられているので，他のジョブの空間を組入れるために移動させることはできない．たとえば10分間の動作の後に([c])，JOB1が終わり，そしてさらに2つのジョブがスケジュールされる．このように図7-5.3は，時間と空間を共有される主記憶装置の三つの場面を示している．静的再配置の効果は主記憶装置の細分化によって悪影響を受ける．**動的再配置システム**では，最も簡単なものでも，細分化を減らせる．

動的再配置では，プログラムのアドレス空間は前もって主記憶装置の物理的なロケーションに結びつけられない．これは，プログラムのアドレスは，プログラムの実行時中CPUの特殊なハードウェアにより**記憶装置参照毎**に変換されるからである．これから，**単一ベース再配置**(single-base relocation)と呼ばれる動的アドレス変換の一つについて述べる．

単一ベース再配置では，プログラムのアドレス空間は，コンパイル，リンク・エディットの後でも常にゼロを起点とする相対的なものである．実行前のロード時に，プログラムのアドレス空間は(余計な変換もせず)単に主記憶装置の十分大きなある連続した領域にロードされる．アドレス空間中のアドレス

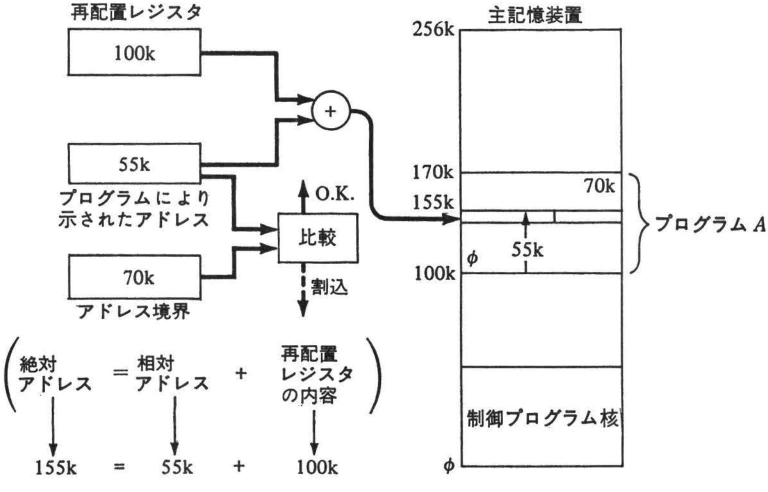


図 7-5.4 再配置レジスタを用いたアドレス変換

は，それゆえ，やはりゼロを起点とする相対的なものである．ローディング時に，コントロール・プログラムはプログラムの記憶装置領域の起点を特別な**再配置レジスタ**に置く．この過程は**図7-5.4**の例で示されている．プログラムAはロケーション100kから始まる主記憶装置中にロードされ，そのアドレスはこのローダによって再配置は全く行なわれない．プログラムAがランし始めると，主記憶装置の100から170kに置かれているにもかかわらず，そのプログラムは0から70kの領域のそのプログラムのアドレス空間への参照を行なう．もしCPUが55kのような論理アドレスをそのまま用いたら，そのプログラムが現在占有していない物理的領域の0から70kの主記憶装置を参照してしまう．そのため明らかに，プログラムAの論理アドレスはCPUによって記憶装置参照ごとに変換されなくてはならない．図で示されるようにこれはハードウェアで行なわれ，主記憶装置中の最終的なロケーションを得るために論理アドレスと再配置レジスタ中の起点アドレスを加える．

システムが多重プログラミングであれば，再配置レジスタにはその時実行されているプログラムの主記憶装置の起点が入っている．他のプログラムに制御を渡す直前に，制御プログラムは，今から実行を始めようとしているプログラムの主記憶装置の起点を再配置レジスタにロードする．制御プログラムのみが再配置レジスタを変更することを許されているので，このレジスタにロードする命令は特権機械命令である．

単一レジスタ再配置を行なうシステムでは，簡単な記憶保護の手法が通常使われる．**アドレス境界レジスタ**(address-bound register)と呼ばれる特別な目的のための第2のレジスタには，プログラム・ロード時に，ロードされるプログラムの一番大きい相対アドレスが制御プログラムによって入れられる．**図7-5.4**の例では，この値は70kである．おのおののアドレス変換の前に(或はしばしばそれと同時に)．論理アドレスはアドレス境界レジスタの内容と比較される．もし論理アドレスの値の方が大きければ，実行中のプログラムがそのアドレス空間の大きさを越えたアドレスを参照しようとしているのだから，保護割込みが生ずる．

単一レジスタ再配置方式を用いると，プログラムは前もって主記憶装置のロケーションに結びつけられてない．主記憶装置の細分化が減りオペレーティング・システムは主記憶装置をより有効に用いる可能性を持てる．**図7-5.3**の[*b*]で示される状況を考えよう．再配置レジスタを持つシステムでは，JOB3のプログラムはJOB1の下の主記憶装置に移動できる．JOB3のプログラムを物理的に動かす代わりに，コントロール・プログラムはプログラムの主記憶

装置中の新しい起点を知り，起点の値の表にこの値を書き移すことだけをすればよい．JOB 3 が再び実行されるとき，主記憶装置のその新しい起点が再配置レジスタに入れられる．JOB 3 が移動されたら，図 7-5.3 で用いられた静的再配置手法によるよりも 5 分も早く JOB 4 の実行を開始することが可能である．

単一レジスタ再配置は**動的再配置**の 1 つの型である．それはアドレス変換が絶え間なく行なわれているのでそう呼ばれる．あるオペレーティング・システムでは再配置レジスタによる方法を拡張し，そしてもう少し巧妙な手法を用いる．おそらくそのような一般化の最も簡単なものでは再配置レジスタ，アドレス境界レジスタの組を数個持ち，1 つはプログラムの，そして他のものはデータの再配置用である．これは非常に簡単なようだが，制御プログラムを精巧にするうえで非常に大きな効果がある．セグメンテーション・システムとページング・システムはその例の 2 つと考えられるが，それらの議論は第 10 章で行なう．

多重プログラミング・システムでジョブをスケジュールするため，制御プログラムはジョブのプログラムを入れるのに充分な主記憶装置を割当てなくてはならず，また(もしあれば)ジョブの要求するすべての I/O 装置とデータ・セットをそのジョブに割当てなければならない．そこで多重プログラミング・システムはジョブ・スケジューリングのため規則(またはその集合)を持たなければならない．その規則は次にローディングや実行を行なうジョブを選ぶアルゴリズムである．スケジューリング規則はジョブにより**要求される**主記憶装置の量や装置によってジョブに与えられるジョブ・クラスなどのパラメータに基づくであろう．現在の多重プログラミング・システムではジョブの待ち行列は通常(ディスクのような)直接アクセス装置に置かれる．ジョブの待ち行列(あるいはその記述子)をかなり速くサーチできるので，種々のスケジューリング・アルゴリズムが使える．ジョブ・スケジューリングの特別な方式のいくつかについては，第 8 章の IBM OS オペレーティング・システムで述べる．

われわれの言うジョブ・スケジューリングは，どのようなアルゴリズムであろうと，主記憶装置にジョブのプログラムを移すが，主記憶装置中ではプログラムはそこにある他のプログラムと CPU の処理を取り合う．つぎの事から**ディスパッチング**と呼ばれる多重プログラミング・システムで次に重要なスケジューリング機能が導びかれる．すなわち，主記憶装置中にあるすべてのプログラムの中から，次にどれに CPU を割当てるべきか．スプーリングのみのシステムでは，リーダ，ライタそして 1 つのユーザ・プログラムがすべてディスパッチされなければならない．もっと一般の多重プログラミング・システムでは

数個のユーザ・プログラムとシステム・プログラムが同時に主記憶装置中にありCPUを奪い合う．そのディスパッチ機能を実現するために，コントロール・プログラムは主記憶装置中に同時に存在するシステム・プログラムとユーザ・プログラムのすべてのリスト(あるいは待ち行列)を持たなければならない．待ち行列には各プログラムを識別するための1つのエントリがある．これらのエントリを**タスク制御ブロック**(task control block)と呼ぶ．この議論のためにタスクをプログラムと同義とする．

主記憶中のどのプログラムに次にCPUを与えるかを決めるのに種々のディスパッチング規則がある．たとえば，ラウンド・ロビン(4章を参照)は1つのディスパッチング規則となりうる．入力リーダや出力ライタのようなシステム・プログラムをユーザ・プログラムと同じディスパッチング規則で制御するかどうかは，一般にシステム設計の選択による．普通にはシステム・プログラムにはより高いディスパッチング優先度が与えられる．ほとんどのIBM多重プログラミング・システムはHSFS (highest-static-priority-first-served) ディスパッチング・アルゴリズムを用いる．それは**横取り的**であり，優先度がより高いプログラムは優先度の低いプログラムに割込みをかけ，CPUの制御を得る．システム・プログラムの多くはディスパッチング優先度が決められていて，システム・ジェネレーション時にのみそれを変えることができる．ユーザ・プログラム(またはジョブ)のディスパッチング優先度は“ユーザ”によって割当てられる．このユーザとはプログラマまたはスケジューリングの責任を負う特別なオペレータ，またはあるシステムでは導入時にその仕事を割当てられた人である．ジョブ・カード，あるいはジョブ-ステップ・カードのJCLカードによって割当てられたディススパッチング優先度は，制御プログラムに伝えられる．ジョブがシステムのジョブ待ち行列に一旦入ると，そのディスパッチング優先度が決まっているので(一般にオペレータはそれを変更できるが)，これはHSFSディスパッチングに等しくなる．ディスパッチング・アルゴリズムが動的に優先度を変える機能を含んでいなければ，一旦，ジョブが実行のためロードされたら，そのディスパッチング優先度は実行中変らない．

HSFSスケジューリングとディスパッチングの考えは，CPU，I/Oチャネル，そして装置を効率よく用いる責任をユーザやオペレータにも負わせている．ユーザのディスパッチング優先度の与え方でこの責任が果される．I/Oで制限されるジョブとCPUで制限されるジョブをうまく混ぜるとよい結果がえられる．たとえば，I/Oで制限されるジョブはI/O装置を動作させておくために一般に比較的少しのCPU時間しか要求しない．そのようなジョブにはCPU

で制限されるジョブに比較してより高いディスパッチング優先度が与えられるべきである．この割当て方針はすべてのシステム要素ができるだけ動作中であるようにさせ，それゆえ使用率とスループットが改善される．

7-6 ソフトウェア・モニタ [AR, BE, CA, CU, CHE, KO, SKI 参照]

システムの動作をモニタすることはデータを集める仕事であり，データが計算機自身の動作で生成されるので，計算機に特に適合した仕事である．特定の手法を論ずる 前に，パフォーマンス・モニタ/測定の目的のいくつかを説明するのがよいであろう．

1. システム設計とデバグの段階でのシステムの開発，改善の援助
2. 新装置やサービスの賃借/買取の決定の助け
3. システムの調整の助け，すなわち，どのシステム・プログラムとテーブルを主記憶中に置くかとか，ディスパッチング間隔等のようなオペレーティング・システムの種々のオプションを選ぶため．
4. システム設計と性能評価の技術・手法を確立する助け．

4章でわれわれはハードウェア・モニタについて触れた．それは装置使用の決定に有効な道具であることがわかり，またポータブルであるという大きな利点を持っていた．すなわち，ほとんどすべてのシステムに対しシステムを変更することなしに同じモニタ用ハードウェアが取りつけられる．しかしハードウェアモニタは大きな欠点を持つ．すなわち種々の動作が，システムおよび問題の，**プログラム**と**データ・セット**の**どこに**集中されているかの詳しい情報が都合よく得られない．モニタされた情報をパーフォーマンスの多くの面を改善するために用いるさい欠くべからざるものである．まさにこの情報がソフトウェア・モニタにより得られる．ソフトウェア・モニタの出力は一般に I/O 装置，記憶領域等への参照回数(あるいは実際の参照の流れ)の表である．この情報は次のように解析される．

1. 目による検査で，(余り実用的でない)
2. 統計のプログラムによる処理によって，
3. モニタされたデータをシステム・シミュレータに入れることによって．

現在，2番目の方法がよく用いられる手法である．しかし方法3についてのいくつかの報告もまた文献に現われている．(たとえば，依頼された会社から

得られるモニタ情報を入力されるIBMのAMAP手法とシミュレータ). シミュレータの大きな利点は, 解析者が現実的な作業負荷の条件のもとにシステム(モデル)の種々のパラメータを使って試すことができることである. タイムシェアリング・システムでこの手法を用いた例が9章にある.

リソースの割当てや使用に影響を及ぼす条件を検知するために, オペレーティング・システムを助けるハードウェア手段の構造からソフトウェア・モニタの可能性が決まる. これらには特権モードと割込み構造が含まれる. そのような合図を与える機構は一般に数がかなり少なく (IBM システム/360 はたった5つの割込み場所を持つだけである), すべての割込みを傍受し, 何がそれらを引き起こしたかの情報をとらえるようにモニタ・プログラムを書くことは本来そんなに難しいことではない. ハードウェア・モニタと異なり, ソフトウェア・モニタは各システムのオペレーティング・システムの構造のかなり詳細な知識を必要とし, あるタイプのシステムから他のタイプへポータブルではない.

割込みの最高発生率は高速なマシンでは非常に高いであろう. これはソフトウェア・モニタに関して次のようなことを暗示している.

1. 割込み事象を把えるためのモニタによる処理が顕著なオーバヘッドとなりうる.
2. モニタリングの時間は割込みが禁止されるので, ソフトウェア・モニタはシステム応答を遅くさせ, また, 特に“リアルタイム”システムでは, タイミングの問題を引き起こすかもしれない.
3. 主記憶中にモニタされた情報を入れておくための場所が用意されなければならない. この場所は割込み率が早いほど大きくなければならない.

最初の2つの問題点は, モニタされた情報のできるだけ少量の即時処理だけ, すなわち, 単にそれを記録する(多分時刻をつけて)ことだけを行なうことにより, かなりの程度解決できる. それからこれらの記録はディスク(あるいは磁気テープ)に書かれ, そしてこの“流れ”はシステムの普通のジョブと同じように走る別のプログラムによって後から処理される. しかし, モニタ時には, ディスク, または, 磁気テープへのレコードのバッファのために主記憶装置内でいくらかの大きさの領域が用意されなければならない. 適当な大きさのバッファ領域により多くの場合, ピークの割込み動作を情報を失うことなく記録できる.

割込み率, マシン速度, モニタ動作に利用できる主記憶の量などに依存して, モニタリング処理自体がシステム速度にかなり影響したり, しなかったりする.

また，非常に詳細なモニタリングは，必要以上の，あるいはある目的に都合よく用いられる以上の情報を生ずることもあろう．これらの欠点はモニタリングの**サンプリング**手法を用いることで解決される．普通，これは，モニタリング・タスクのためにインタバル・タイマによってマシンが割込まれるようにして規則的な時間間隔でのみモニタリング機能を起動することを指す．これらの時刻にモニタ・プログラムは，リソース，プログラム，データ・セット参照の動作のレコードを含むようなテーブルを調べ，そしてモニタ・レコードを書き，モニタ動作表を更新する．応答に用いられるいくつかの情報は常にモニタ・サンプル回数より頻繁に記録されていることに注意しなければならない．しかし，これらの記録自体は時間-空間を使う割込みを必要としない．注意してオペレーティング・システムを設計すると，記録は簡単にすばやく行なわれうる．それで時間をくうモニタ・レコードは比較的稀なサンプリング時刻に書かれるだけである．サンプリング間隔の長さはシステム・プログラマやオペレータによって指定されるパラメータとして簡単に与えられるであろう．たいてい，間隔は大きな値にセットされ，その結果モニタリングのオーバヘッドが減り，生の(しかしながら有益な)性能の情報が得られる．間隔はより詳細な検討のため短くされることもある．

Cantrell と Ellison［文献 CA］はサンプリング・ソフトウェア・モニタで得られた代表的な結果について報告している．それらの内容は GE-635 計算機(約 10^6 命令/秒）の大きなオペレーティング・システム開発で性能改善に役立った．そのモニタの1つは1ミリ秒間隔で CPU の状態をサンプルした．それは CPU を 30% 遅くしたが，いくつかの性能バグを見つけるのに役立った．バグの1つは FORTRAN コンパイラにありその性能を 27% 悪くしていた．もう1つのモニタは(標準) 2秒間隔でリソース全体とジョブの状態をサンプルし，チャネルと CPU の使用率，プログラム開始・終了時刻などの情報を記録した．それは主記憶装置 4k 語を必要とし，約 10% のシステム・オーバヘッドを要した．

IBM OS/360 システムで用意されたモニタリング機能のいくつかについての議論は 8.5 節を参照されたい．

7-7 順序処理の協調：デッドロック理論［COF, DI, HAB, HO 参照］

リソースを共有し，しかも並行動作を許すシステムは**デッドロック**と呼ばれる“論理的病い”にさらされている．リソース要求の様子がいくつかのジョブ

が決して終了しないような関係になっているときデッドロックが起こる．この節では，デッドロックのいろいろなクラスを明確にするように工夫された簡単な例を主として用いて，この問題の一般的な原理を論ずる．次の章の6節で，特定のオペレーティング・システム，IBM OS/360 MVTにおいてデッドロック取扱いのため用意されたものについて論ずる．

動作中のシステムに対するわれわれの一般的な概念はジョブの集まりであり，すべてのジョブは互いに論理的に独立であり，各ジョブはいろいろなリソースを時々要求する．注目されたジョブはどれでも，リソース割当て，返却の要求が書かれたシーケンスと考えられるであろう．われわれはそのようなシーケンスを**プロセス**と呼ぶ．プロセスはステートメントのシーケンスからできあがっているという点で，書かれたプログラムに似ているが，各プロセス・ステートメントはリソースの割当て，返却によりジョブの状態の変化をもたらす．(リソースにはCPU，チャネル，ディスクや磁気テープの駆動装置，主記憶や2次記憶の領域，共通プログラムなどを含む.)ここではリソース管理のみを考えるので，プロセスは普通の意味でデータを操作するプログラム・ステートメントを除いてもよい．

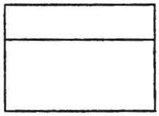
計算機のリソースの多くは，少なくとも異なった時刻には，そのユーザ間で共有される．明らかに，もし(意図的であろうと偶然であろうと)あるジョブが無限に長い間，あるリソースの制御を独占する状況が起これば，他のジョブへリソースを与えず，それらのジョブを無限に遅らせるであろう．そのような“締出し”現象は種々の状況のもとに起こり得て，それらの状況のあるものは簡単に回避できるが，あるものは非常に複雑で解きにくく，これはシステム構成と設計での基本的な決定に帰因している．

理論的解はないが満足できる実際的な解がある締出し状況の例は，“エンドレス・ループ”問題である．計算可能性の理論により，有限時間で終了するすべてのプログラムのクラスを考えると，あるプログラムがこのクラスの要素かどうかを判断するプログラムはそれ自体そのクラスの要素でないことがわかる．簡単に言えば，これはエンドレス・ループのあるすべてのプログラム中のエンドレス・ループを見つけだす厳密な手続きを考えることは不可能なことを意味しており，それゆえ，長時間走っているプログラムが正当に長く，最後には終了するのか，あるいは誤っていて決して終らないのかを知る正確な一般的方法をわれわれは持っていない．しかし，CPUを独占する問題に対する非常に納得のゆく実際的な答えがあるので，理論から厳密な解が得られないといって落胆することはない．これは各ジョブの“実行時間”の見積りを（たとえば

ジョブ・カードで)ユーザが宣言することを要求する方法(多くのシステムで用いられている) である．もしジョブが実際使用している CPU 時間がこの見積りを超えたら，そのジョブはシステムによって終了させられる．ユーザは以下のような賞罰規則によりかなりうまくジョブ時間を見積るようになる；もし見積りが長すぎると，多くのスケジューラは長いジョブに低い優先度を与えるので(4章参照)，ジョブは往々にして長いスケジューリング遅れを被る．一方もし見積りが短かすぎると，見積られた時間を超え，そのジョブは途中で終了されるであろう．(しかし，それでもなお，その時間には課金される．)また，後者には，新たな実行をさせるのに遅れがあるだろう．このように，CPU 独占を避ける戦略の全容はユーザが与える情報と自動的な応答を混ぜ合せたものである．これは CPU リソースが**割込み可能である**という事実によっており，それでこのリソースを使うジョブは横取りされうる．

並行処理可能なシステムで起こる1つの状況はリソースあるいはプロセスの**排他的使用則**の強制である．たとえば，すべての座席予約を記録している FILE と呼ばれる共通データに，2つの CPU がアクセスするような飛行機座席予約システムを考えよう．さてほぼ同時に2つの代理店が同じ便の座席を要求し，しかもただ1つの座席しか空いていないとしよう．もし排他的(一時に1つ) アクセスが強制されなければ，両方の代理店に座席が割当てられる可能性がある．さてこの例で各代理店は異なる CPU の中のプログラムによりサービスされ，両方の要求が同時に処理され得ると仮定する．**図 7-7.1** は一見，FILE への排他的アクセスを保証する方式に思えるものである．それは確かにある場合には働くが，排他的アクセスをし損じる場合が示されているように，排他的アクセスを**保証**できない．アクセス制御の論理は制御変数 X をプログラム・スイッチとして用いる．X は2つの値，BUSY と IDLE のうちの1つを持つ．FILE にアクセス可能な2つのプロセス，$P1$ と $P2$ は X にアクセスし，それが BUSY かどうか調べる前処理を行なうことからそれぞれのプロセスを開始する．もし X が BUSY であるとわかると，プロセスは再び X を調べに戻り，(FILE を現在使っているプロセスにより)X が IDLE にセットされるまでそれを続ける．制御を誤まる場合が図に示されている．基本的に，誤りは初めのプロセスが制御の前処理を終える前に，2番目のプロセス，$P2$，が前処理を開始することに原因がある．$P1$ が X を調べ，しかしまだ X を BUSY に変える前に，$P2$ は X がまだ IDLE にセットされているのを知って，そのため $P2$ も同じことを行なう．それゆえ，$P1$ と $P2$ は共に FILE を同時にアクセスし，一時に1つしか使用できないという規則を破る．この問題の解決

制御変数 X



制御変数の規定

X ＝'IDLE' または 'BUSY'

プロセス $P1$	プロセス $P2$
(1)　GET X→T1	(1)　GET X→T2
(2)　IF T1='BUSY'GO TO(1)	(2)　IF T2='BUSY'GO TO(1)
(3)　T1←'BUSY'	(3)　T2←'BUSY'
(4)　PUT T1→X	(4)　PUT T2→X
(5)　($P1$がFILEをアクセスする動作部分)	(5)　($P2$がFILEをアクセスする動作部分)
(100)X←'IDLE'	(50) X←'IDLE'
(101)END $P1$	(51) END $P2$

条件：一時に高々一つのプロセスのみ（その動作部分で）FILEにアクセスが許される．二つのプロセッサ(CPU)があり，一つが$P1$を実行しているとき同時にもう一つが$P2$を実行できる．すべてのステートメントは同一時間で実行される．

タイミング・シーケンス例

時間 →	1	2	3	4	5	6	7	
$P1$ステップ	(1)	(2)	(3)	(4)	(5)	(6)	(7)	等 →
$P2$ステップ	-	(1)	(2)	(3)	(4)	(5)	(6)	

結果：$P1$と$P2$の二つのプロセスはそれらの動作部分で同時にFILEをアクセスする

図 7-7.1　排他制御の失敗例

法は少なくともプロセスのある制御部分については**排他使用を強いる**ことである．たとえば，もしステップ(1)から(4)の間はただ1つのプロセスしか制御変数 X にアクセスできないなら，問題は解決するであろう．この状況がまさしく共通の主記憶にアクセスするマルチプロセッサ・システムで起こる．IBM システム/360/370 では，TEST-AND-SET と呼ばれる特別な命令が用意されており，それは次のように実行される．その命令は任意のバイト X の番地を指定し，それからその番地がテストされ，そして CPU のコンディション・コードを以下のようにセットする．

X の左端のビットが 0 のとき 0

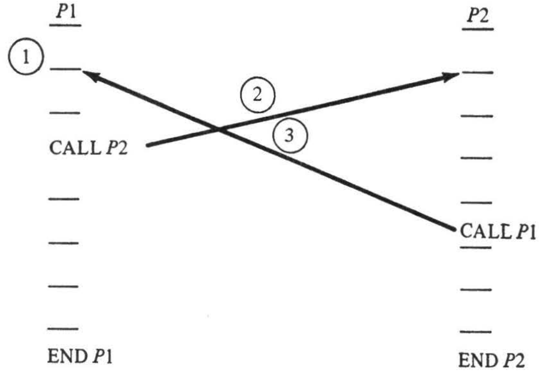
X の左端のビットが 1 のとき 1

そしてバイト X はすべて 1 にセットされ，このすべての処理の間，バイト X

への他のアクセスは禁止される．TEST-AND-SET 命令は多くのコンピュータ・システムで見られるアクセス環境での主記憶装置内の領域に対する排他的アクセス制御を保証するのに欠くことのできないハードウェア構造である．

次に図 7-7.2 に示されるように 2 つのプロセスが互いに際限なく進行を妨げあうような論理的状況を考えよう．1 つのプロセスが他のプロセスをコールしたら，コールしたプロセスはコールされたプロセスがその END に達するまで，コールした点で止まったままでいると仮定する．また，一旦コールされたら，そのプロセスが END に達するまで他のコールは実行されないとする．後者の排他使用の規則は前に述べた TEST-AND-SET のような機構によって実施されるであろう．プロセス $P1$, $P2$ が最初共にアイドルであり，（図には書かれていない）他のあるプロセスが $P1$ をコールした時点から図 7-7.2 の解析を始める．見ての通り，$P1$ はそのうち $P2$ をコールし，その時点で $P1$ は $P2$ が END に達するのを待ちつづける．そして $P2$ は $P1$ をコールする点に達するまでそのシーケンスを進める．$P2$ は $P1$ がその END に達するまでそこに立ち止まる．しかし簡単にわかるように，$P2$ による $P1$ のコールは開始することすらできないので，この待ちは無駄である．$P1$ はすでに実行中であり $P2$ の END を待っているので決して終らない．言い換えると，プロセス $P1$ と $P2$ は互いに進行を妨げ永久に待ちつづける運命の“死の抱擁”にある．

多くのシステムでは，リソースは**順次再利用可能**であり，それは簡単には，一旦あるリソースがプロセスに割当てられたら，そのプロセスがそれを手離す



（注：丸で囲まれた数字は相対的な時刻を示す）

図 7-7.2 簡単なデッドロック状況

まで他のプロセスはそのリソースを使用できないという事である．これは正に上で述べた排他使用の別の言い方である．以下の議論では，われわれは(特にことわらないかぎり)順々に再利用可能なリソースのみを仮定する．

上のデッドロックの例は互いにコールし合うプロセスを含んでいた．しかしデッドロックは互いにコールし合わなくとも起りうる．**図7-7.3**には互いにコールし合わないが，共通のリソース($R1$と$R2$)にアクセスする2つのプロセスの例を示す．$P1$と$P2$は両方とも2つのリソースを割当てる(要求する)が逆の順番で割当てられる点に注意しよう．この例では，$R1$がまず最初に$P1$に割当てられ，そして$R2$が$P2$に割当てられるような時間関係を仮定しよう．次に$P1$は$R2$を割当ててもらおうとするが，$R2$はすでに使用中であるので$P1$は立止まる．少し遅れて，$P2$は$R1$を割当ててもらおうとするが，$R1$は$P1$によって使用中である．この時点で$P1$と$P2$の両方とも，リソースの返却を待って止まっているが，見てわかるように，2つのプロセスはRELEASE文まで進めないので，リソースの返却は決して起こりえない．そ

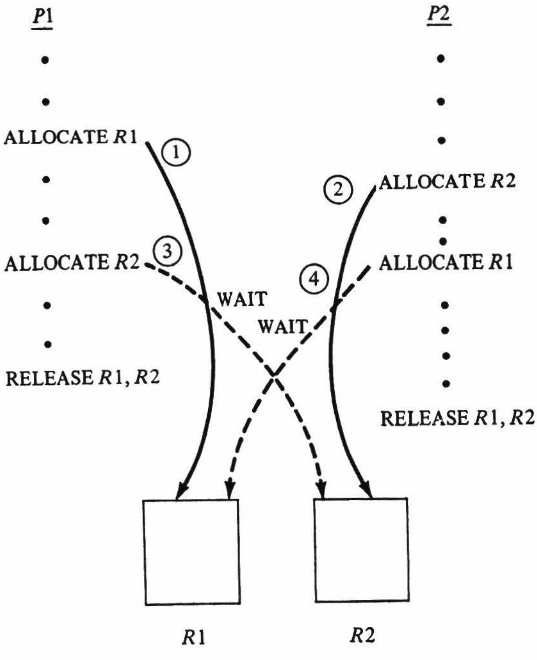


図 7-7.3　共通リソースによるデッドロック

れでこれはデッドロックのもう1つの例となる.

これから後をよく理解するために**タスク**という語を導入する．われわれのデッドロックの議論の流れでは，タスクはプロセスの実行形式である．言い換えれば，タスクとプロセスの関係は実行プログラムと書かれたプログラムの関係に等しい．タスクをプロセスの実行ステップのレコードと考えてもよい．この基本的な定義を使って，デッドロック理論のより形式的な側面を与えよう.

デッドロック問題の区分とそれをどのように扱うかについて多くの理論的研究が最近なされた． Coffman, Elpnick, Shoshani (文献 COF) は以下の条件のすべてが起こるとデッドロックになるとまとめた.

1. **相互排除**　タスクはそれが必要とするリソースの排他制御を要求する.
2. **待ち**　タスクは他のリソースを待つ間そのタスクにすでに割当てられたリソースを持っている.
3. **横取りなし**　リソースはタスクがそれを返却しない限りタスクから撤去されえない.
4. **循環待ち**　各タスクは連鎖の次のタスクによって要求される1つ以上のリソースを持っている.

デッドロック問題で役に立つ形式的なモデルは次のようなものである．実行時のある任意の状態にあるタスクの集合を $\{T_1, T_2, \cdots, T_n\}$ とする． 相異なるリソースの集合 $\{R_1, R_2, \cdots, R_m\}$ を考え， 各タイプのリソースはただ1つであると仮定しよう．このモデルのどの場面でも，システムの状態は(**図 7-7.4** にあるような)グラフで表現される．節点はリソースを表わし, R_i から R_j への枝は考察している時点で, あるタスクが R_i を保有していて R_j を要求していることを表わす. もし上で述べた条件の初めの3つが成立すると, 循環待ちの存在する必要十分条件はグラフが閉ループを持つこと，すなわちある節点から枝をたどってその節点自身に戻る経路が存在することであることがわかるだろう.

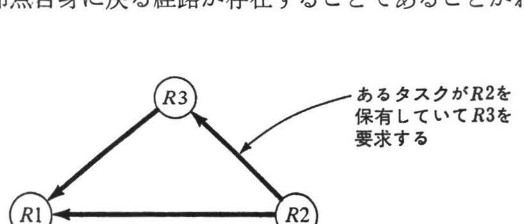


図 7-7.4　グラフ・モデル

デッドロック問題を処理するための方策は3つの型に分けられるすなわち,

1. 検出
2. 予防
3. 回避

デッドロックの**検出**は,デッドロックの状況が存在するかを検出するため周期的にリソース割当ての状態と満足されていない要求を調べるアルゴリズム(すなわち,システム・プログラム)によって行なわれる.もし検出されたら,システムはデッドロックに含まれるいくつかのリソースを横取りし,それゆえリソースを要求しているプロセスのいくつかを異常な形で終わらせる.検出プロセスはほとんどの場合デッドロックがないことを見つけるので,検出のためのプロセスがオーバヘッドになる欠点がある.また,デッドロックがもし仮に存在していたとしても,デッドロックが始まってからそれが検出されるまでの時間は無駄である.結局,検出策は検出後のプロセスを横取りする能力にかかっている.これはある装置(磁気テープ駆動装置のような)の場合難しい.

デッドロック**予防**策はある順序付けの規則でプロセスの構造を制御することによる.それに従えば,デッドロックは決して起こり得ない.予防策は前に述べたデッドロックが起こるためにすべてが揃わなければならない4つの条件に基づいたものであろう.だから,たとえ4つの条件のうち1つでも起こらないことが保証できれば,デッドロックは起こらない.たとえば,システム構成が前3つの条件を許すが,同じリソースを要求する2つのプロセスは同じ順番でそれらのリソースを要求しなければならないとすることで循環待ち条件を避けるとしよう.(前に示したように,逆の順序で要求すると,循環待ちが生じる.)今や循環待ちが不可能となったので,デッドロックも不可能となる.またこれは横取りや検出の機構なしで行なわれる. IBM OS システムの設計ではこの方法がとられた.(内容は8.6節を参照)

デッドロック**回避**策のシステム・モデルとそれに伴なうアルゴリズムは Habermann によって示された.われわれの場合は次のような標記法を用いる.

MR = システムにある各種のリソースの数のベクトル

例: $MR = 5, 2, 1, 2, 1$ はシステムが5つのテープ駆動装置,2つのディスク駆動装置. 1つの CPU, 2つのプリンタ,1つのカード読取機をもつことを意味する.

NR = リソースの種類の数(MR の次元)

NT = 現在動いているタスクの数 (開始され終了していないタスクの

数）

$MC =$ 各タスクのリソース各種類への**最大要求**の行列

$MC[I;\ J] = MC$ の要素 I, J はタスク I が要求できる種類 J のリソースの最大数である.

$RR =$ 現在使用可能な，すなわちどのタスクにも割当てられていない各種類のリソースの数のベクトル

$\mathrm{CC} =$ 各タスクのリソース各種類への**現在の要求**の行列

RR の要素の値は次式で与えられることは容易にわかる.

$$RR[J] = MR[J] - \sum_{I=1}^{NT} CC[I,\ J] \qquad J = 1,\ 2,\ \cdots,\ NR$$

このモデルの大きな特徴は各タスクが開始するときリソースの各種類に対する**起きうる要求の最大値**を宣言しなければならないことである．そしてそれは MC 行列の 1 つの行として置かれる．しかしこの最大値はただちにそれだけの数のリソースがタスクに予約されることには**ならない**．そうした予約は無駄の多いリソース割当てとなるだろう．そのかわり，各タスクはリソースが必要となったとき要求を出すことが許される．もちろん，その合計は宣言した各種類の最大値を越えることはできず，また，要求を受付ける前に，コントロール・プログラムは，要求を出しているタスクにリソースを与えてもまだ十分なリソースが残っていて，すべてのタスクが何らかの順序で終了できるということを確かめねばならない．すべてのタスクが終了するというこの保証がデッドロック回避問題の中核であり，そしてわれわれは今からどうすればこれが可能かを見てみよう．

まず 2 つの関連する語を定義しよう． システムの**状態**とは行列 MR, MC, CC, RR の値のある集まりである．**安全な状態**とは，すべての実行中のタスクが宣言したリソースの最大必要数を獲得でき，したがって，終了できるシーケンスが存在する状態として定義される． ささいな例ではあるが重要な例として， すべてのタスクが最大必要数を宣言した(しかし， どれも要求はしていない)直後には，システム状態は確かに安全である．なぜならすべてのタスクは，あるタスクが終了してから他のタスクがスタートするという厳密な順序で走らせてもよく，そのとき終了できるからである．もっと一般的には，状態が安全かどうか調べるには，われわれはまず，すべての実行中のタスクがその宣言した最大のリソースをすべて得ることが許され，確実に終ることができるのに充分大きな値を RR が持つかどうかを尋ねる．形式的には，これは

$$RR[J] \geqslant MC[K;\ J] - CC[K, J] \qquad J = 1, 2, \cdots, NR$$

なるタスク K が存在するかどうかをテストすることによってなされるであろう．もしそのような K が見つからなければ，明らかにデッドロックの可能性がある．しかし，この関係を満たす少なくとも1つのタスク K があれば，このタスクは終わることができる．それが終った時，それは保持しているリソースを返却する．これらのリソースは試験用 RR ベクトルを都合よく変更し，恐らく他のタスクが終ることを許すであろう．それでこの新しい試験用 RR ベクトルを用いてテストを続ける．原則的には（しかしすぐ後でわかるように実際的ではないが）タスクの終了のすべての可能なシーケンスについて 結果としてすべてのタスクが終了するようなあるシーケンスがあるかを調べることができる．そのようなシーケンスがあれば初めの状態は安全である（もしそのようなシーケンスがなければ安全ではない）．

デッドロックの回避を保証するためには，システムの状態は常に安全な状態でなければならないことを示すことができる．さてわれわれは安全な状態を保証するアルゴリズムの概略を述べよう．システムが安全な状態にあり，あるタスク I が今使っているのに加えてリソースをベクトル R として要求したとしよう．もし要求が認められたら，システムは**試行状態**(trial state)と呼ばれる新しい状態に置かれる．今中心となる質問は“その試行状態は安全な状態か.”である．この答えは上で述べた試験法で得られるであろう．

もっと効率的なアルゴリズムは以下のようである．試行状態の使用可能なリソース($RR - R$)から始めて，タスク I が終了できるかどうかをテストする．もし終了できるなら，試行状態は安全な状態で，テストは終り，I の要求は認められることができる．もし終了できなければ，他のタスクが終了できるかを調べる．他のタスクがどれも終了できなければ，テストは終り，要求は認められない．もしあるタスクが終了できれば，そのタスクが RR に返却するリソースを加えて，もう一度タスク I が終了できるかどうかを調べる．もし終了できるなら試行状態は安全であり，もしそうでなければ，テストを続ける．要求を出しているタスク I が終了できることがわかった時点で試行状態が安全であることが保証されることに注意しよう．なぜそうなるかは，試行状態の前の状態は安全であったからすべてのタスクが終了できたということを想い起こせばよい．したがって，現在の要求が認められた後でタスク I が終了できることを示しさえすれば事足りる．タスク I が終了した時には，そのすべてのリソースを返却し，それで，少なくとも試行状態の前の安全な状態と同じ程度都合のよい

条件に RR を置く．

例として次のようなシステム状態を考えよう．

全リソース：　$MR = 5, 5, 5, 5$

タスク数：　$MT = 3$

最大要求数：	MC	現在要求数：	CC
タスク 1	2 2 3 3		1 1 2 1
〃 2	1 1 0 1		1 1 0 1
〃 3	3 1 2 2		2 1 2 2

残っているリソースは

$$RR = 1, 2, 1, 1$$

となる．タスク 1 が新たにリソース

$$R = 1, 1, 1, 1$$

を要求したとしよう．この要求が認められるかどうか調べるために，まず，もし要求が認められたら CC と RR の行列はどうなるか(試行状態)を計算する．

$$CCT = \begin{matrix} 2\ 2\ 3\ 2 \\ 1\ 1\ 0\ 1 \\ 2\ 1\ 2\ 2 \end{matrix}$$

$$RRT = 0, 1, 0, 0$$

これからわれわれはベクトル和 ($RRT + CCT$ の第 1 行) と MC の第 1 行を比較して前者は 4 番目の種類のリソースを除いて十分な値であることがわかる．それでただちにタスク 1 の終了を保証するのに十分なリソースはない．しかし(まだ)われわれは絶望しない．われわれは同様の比較をタスク 2 について行なう．その結果は好ましく，タスク 2 は終了できることを保証していることがわかる．タスク 2 が終了すると仮定すると，MC 行列の第 2 列を見るとタスク 2 が 1, 1, 0, 1 の量のリソースを返却し，それで RRT の値が同じ数だけ増えることがわかる．それから RRT の新しい値を使ってタスク 1 が終了できるかどうかを調べる新たなテストを行なう．こんどは，タスク 1 が終了できることがわかる．それゆえ，試行状態は安全で，タスク 1 のリソース要求は認められる．

われわれは検出，予防策を簡単に，また回避策をある程度詳細に議論した．おのおのの方策を混合したものがオペレーティング・システムを構成する論理

階層のいろいろなレベルで提案されている(文献 HOW 参照).

演　習　問　題

1. プログラムの記号的な名前空間とそのアドレス空間(オブジェクト・モジュール)との差異について記せ.

2. オペレーティング・システムのない環境下でのオープン・ショップ・スケジューリングとクローズド・ショップ・スケジューリングの特徴について述べよ.

3. “ブートストラップ”ローダとは何か？　また，なぜそれが必要か.

4. 次のものをコンパイル時(それより後でなく)に結合した結果の利点，欠点は何か？

(a)　名前とアドレス

(b)　I/O 装置名と I/O 装置

5. **結合時**と**アドレス空間**の概念を用いて，以下のものを比較・検討せよ.

(a)　アブソリュート・プログラム

(b)　静的再配置

(c)　オブジェクト・モジュール

(d)　ロード・モジュール

(e)　リンケージ・エディタ(システム・プログラム)

(f)　ローダ(システム・プログラム)

6. ジョブ制御言語の基本的な機能は何か.

7. 単一プログラム・オペレーティング・システムにおけるジョブ入力ストリームとは何か. 一般的にそれはシステムのどこにあるか.

8. 単一プログラム・システムでのジョブ・スケジューリングとして一般にはどのようなスケジューリング・アルゴリズムが用いられるか.

9. 単一プログラム・オペレーティング・システムにおける制御プログラム核，そしてジョブ・スケジューラ・プログラムの基本的な機能は何か.

10. 単一プログラム・システムのシステム常駐は何か. またそれは通常どこにあるか.

11. JCL 手続きライブラリとは何か. またその機能は何か.

12. 単一プログラム・オペレーティング・システムでのIPL過程ではどのようなことが行なわれるかについて記せ.

13. リンケージ・エディタにおける2つの基本的機能とは何か. 結合時とアドレス空間の点から問に答えよ.

14. スプーリング・システムの基本的な目的は何か.

15. (a) スプーリング・システムにおける**システム・リーダ**, **システム・ライタ**との機能について述べよ.
(b) スプーリング・システムのジョブ入力ストリームは一般にどこにあるか?
(c) スプーリング・システムで有効なジョブ・スケジューリング・アルゴリズムであって, 問題プログラムの実行がシステムI/Oと完全にシーケンシャルに行なわれるシステムでは実用的でないアルゴリズムとはどのようなものか.
(d) スプーリングの下で一般的にCPUはどのようにスケジュール(ディスパッチ)されるかを記せ.

16. 単一プログラム・システムに対してマルチプログラム・システムのもつ**利点と欠点**を次の観点から述べよ.
(a) 個々のシステム・ユーザ
(b) 導入・設置の管理者

17. 単一プログラム・システムおよび, マルチプログラム・システムにおいて, 次の事項の必要性について簡単に述べよ.
(a) リアル・タイム・クロック
(b) インターバル・タイマ

18. マルチプログラミング・システムにおける記憶保護について次の点を簡単に述べよ.
(a) なぜそれが必要とされるのか.
(b) 主記憶装置の保護は一般にハードウェアによって与えられるのか. それはなぜであるか.
(c) ディスクの保護は一般にハードウェアによって与えられるのか. それはなぜであるか.
(d) IBMシステム/360ではどのように主記憶装置保護が行なわれているか.

(e) ハードウェアによる主記憶装置保護の他の型の方法を少なくとも1つ示し，その利点，欠点を比較・検討せよ．

19. (a) マルチプログラミング・システムではなぜ特権命令が用意されていることが多いのか．

(b) CDC-6600 システムでは特権 CPU 命令がない．文献からこのシステムを調べ，同様のことをどのようにして行なっているかを説明せよ．

20. スーパバイザ・コール(SVC)割込みの目的は何か．

21. 割込み方式でどのようにして横取り方式の CPU ディスパッチングを行なうか．

22. 静的(ロード時)プログラム再配置のシステムではどのような主記憶の細分化が現われるかを述べよ．

23. (a) 動的再配置の行なわれるマルチプログラミング・システムではいつプログラム再配置が行なわれるか．

(b) どのような特別なハードウェアが要求されるか(少なくとも1つの方式を記せ)

(c) 静的再配置では得られず，動的再配置で得られる主記憶装置割当ての融通性とは何か．

24. マルチプログラミング・システムのジョブ・スケジューリングにはどのような機能が含まれているか．

25. (a) ソフトウェア・モニタとハードウェア・モニタから得られる情報の種類を比較せよ．

(b) ソフトウェア・モニタの過程自身が性能に及ぼす影響と，そのような影響を最小にする方法について論ぜよ．

26. 無限ループは計算機システムでは根本的な問題点である．それに対処する実用的な手法はどのようなもので，なぜ用いられるか述べよ．

27. 多くのマルチプロセッサ・システムで TEST-AND-SET 命令が見受けられる．その目的について論ぜよ．

28. 順次再利用可能なリソースとは何か．

29. デッドロックを起こすための4つの条件を列挙せよ．

30. デッドロックに対処するための次の方法の利点・欠点を比較せよ.
　（a） 予防
　（b） 検知
　（c） 回避

31. （a） 7-7節で詳細に論じたデッドロック回避アルゴリズムをシミュレートするプログラムを書け.
　（b） そのアルゴリズムがデッドロック回避の能力をもつことを示すための少なくとも3つの例を示せ.（例は単にシステムに接続されているリソース以上のリソースを要求するものであってはならない.）

32. 動的再配置をするマチルプログラミング構成と，それをしないマルチプログラミング構成でのいくつかのジョブ・スケジューリング規則とディスパッチング規則を，シミュレーションによって，調査し評価するためにこの計画課題が作られた．システムの目的は最大スループットである．下記に示すシステムのシミュレータを書きなさい．モデル化されるシステム・リソースはCPU 1台，I/Oリソース1つ，それに300kバイトの容量を持つ主記憶装置である．作業負荷モデルは各ジョブを次の4つの数の入力で特徴づける.

CPU要求時間，C
主記憶装置要求量，S
I/O要求の平均時間間隔，R
I/O要求の平均継続時間，D
（これらのソース・データとしてD部を見よ.）

任意のジョブは任意の時刻に6つの状態の1つの状態にある．状態遷移は次の表で示される．（その遷移をわかり易くするために状態図を書くとよいだろう.）

現在と状態	次の状態	遷移を引き起こす事象
INITIAL	READY	ジョブ・スケジューラによるジョブの選択
READY	CPUTIME	ディスパッチャによるジョブの選択
CPUTIME	JOBEND	ジョブの終了
CPUTIME	IOQ	I/O要求
IOQ	IO	I/Oが利用可能
IO	READY	I/Oの終了

（a）　調べるべきジョブ・スケジューリング規則

クラス S　静的再配置のみ

S. 1 - FCFS(ジョブの到着順)

S. 2 - 最小(主記憶要求)ジョブを最初に

クラス R　動的再配置

R. 1，R. 2 - S1，S2 と同じ

R. 3 - 解答者自身の規則を決めよ．

（b）　ディスパッチング規則

D. 1 -(READY 状態のものの中で)最初に主記憶装置に入ったジョブ

D. 2 - 累積 I/O 時間の最も大きなジョブ

D. 3 - 累積 CPU 時間の最も少ないジョブ

D. 4 - 解答者自身の規則を決めよ．

（c）　データは理想的には実際の計算機システムから取られるべきである．テストの目的には，2章の問8，10の記憶容量，時間のデータを使いなさい．ここで時間は C，すなわち CPU 時間とする．R と D は次の平均をもつ指数分布として生成せよ．(上の記法を見よ)

$$R = .1C,\ D = .2R$$

（d）　シミュレータは少なくとも次の出力を出さなくてはならない．

（1）　スループット，平均経過時間

（2）　主記憶装置利用率(平均記憶容量-時間積)

（3）　CPU 利用率

（4）　I/O 利用率

（e）　シミュレータ・ロジック，その有効性をどのようにテストしたか，上述の場合の結果，そして解答者が有用と思う事柄についてのレポートを作成せよ．

参　考　文　献

1. [AR] Arden, B. and D. Boettner: Measurement and Performance of a Multiprogramming System, *Second ACM Symposium on Operating System Principles*, Princeton Univ., 1969.
2. [BE] Bemer, R.W. and A.L. Ellison: Software Instrumentation Systems for Optimum Performance, *Proc. IFIP Congress 68*, pp. C39-C42, August, 1968.
3. [CM] Campbell, D.J. and W.J. Hefner: Measurement and Analysis of Large Operating Systems During System Development, *AFIPS Proc.*, vol. 33, part 1, 1968 FJCC.
4. [CA] Cantrell, H.N. and A.L. Ellison: Multiprogramming System Performance Measurement and Analysis, *AFIPS Conf. Proc.*, vol 30, pp. 213-221, 1968 SJCC.

5. [CHE] Cheng, P.S.: Trace-Driven System Modelling, *IBM Systems Journal*, vol. 8, no. 4, p. 280, 1969.
6. [COF] Coffman, E.G., M.S. Elphick, and A. Shoshani: System Deadlocks, *Computing Surveys*, vol. 3, no. 2, June, 1972.
7. [DE] Denning, P.J.: Third Generation Operating Systems, *ACM Computing Surveys*, vol. 4, no. 1, 1971.
8. [DI] Dijkstra, E.W.: The Structure of the "THE" Multiprogramming System, *Comm. ACM*, vol. 11, no. 5, pp. 341-346, May, 1968.
9. [DO] Donovan, J.J.: "Systems Programming," McGraw-Hill Book Company, New York, 1972.
10. [FR] Freeman, D.N. and R.R. Pearson: Efficiency vs. Responsiveness in a Computer Utility, *AFIPS Proc. SJCC*, 1968.
11. [GO] Gosden, J.A. and R.L. Sisson: Standardized Comparisons of Computer Performance, *Information Processing 1962 Proc. IFIP Congress 62*, North-Holland Publishing Company, Amsterdam, 1962.
12. [HAB] Habermann, A.N.: Prevention of System Deadlocks, *Comm. ACM*, vol. 12, no. 7, pp. 373-377, 385, July, 1969.
13. [HO] Holt, R.C.: Some Deadlock Properties of Computer Systems, *ACM Computing Surveys*, vol. 4, no. 3, September, 1972.
14. [HOW] Howard, J.H., Jr.: Mixed Solutions to the Deadlock Problem, *Comm. ACM*, vol. 6, no. 3, pp. 427-430, July, 1973.
15. [HU1] Hume, J.N.P. and C.B. Rolphson: Scheduling for Fast Turnaround in Job-at-a-time Processing, *Proc. IFIP Congress*, 1968.
16. [HU2] Huseman, L.R. and R.P. Goldberg: Evaluating Computer Systems Through Simulation, *Computer J.*, vol 10, no. 2, pp. 150-156, August, 1967.
17. [HU3] Hutchinson, G.K.: A Computer Simulation Project, *Comm. ACM*, vol. 8, no. 9, pp. 559-568, September, 1965.
18. [KO] Kolence, K.W.: A Software View of Measurement Tools, *Datamation*, January, 1971.
19. [LA] Lampson, B.W.: Dynamic Protection Structures, *AFIPS Proc.*, 1969 FJCC.
20. [PR] Presser, L. and J.R. White: Linkers and Loaders, *ACM Computing Surveys*, vol. 4, no. 3, September, 1972.
21. [SAY] Sayers, P. (ed): "Operating Systems Survey," Auerbach Publ., 1971.
22. [SE1] Seaman, P.H. and R.C. Soucy: Simulating Operating Systems, *IBM Systems Journal*, vol. 8, no. 4, pp. 264-279, 1969.
23. [SED] Sedgewick, R., R. Stone, and J.W. McDonald: SPY-A Program to Monitor OS/360, *AFIPS Conference Proc.*, vol. 37, pp. 119-128, 1970.
24. [ST1] Stanley, W.I.: Measurement of System Operational Statistics, *IBM Systems Journal*, vol. 8, no. 4, pp. 299-308.
25. [TS] Tsujigado, M.: Multiprogramming, Swapping and Program Residence Priority in the FACOM 230-60, *AFIPS Proc.*, vol. 32, pp. 223-228, 1968.

第8章

OS/360 オペレーティング・システム

1964年にIBM社は内部(CPU)速度の範囲が1から50の，小型から中大型までの計算機の6つのモデルからなる系列，システム/360を発表した．これらのシステムの大きな目的は**プログラム互換性**であり，これはすべてのモデルが同じ機械語のプログラムを走らせることができるということである．これを可能とする手始めに，すべてのモデルに一様のマシン・**アーキテクチャ**が用いられた．すなわち，すべてのモデルは同じデータ表現，命令セット，アドレス方式，入出力ロジックを持っている．これらの特徴が一様であることは，プログラム互換性を保証する必要条件ではあるが，十分条件ではない．それ以外に要求される条件はすべてのモデルが同じ量のいろいろなシステム・リソース，特に主記憶装置に対する論理的アクセスを持つことである．この条件はシステム/360では満足されておらず，この理由のため多くのシステム・ソフトウェア・プログラムについていろいろなバージョンが供給された．たとえば異なる主記憶装置の大きさに対してFORTRAN F, G, Hコンパイラが供給された．システム/360のシステムBのリソースが少なくともシステムAのリソースと同じ量なければ，ユーザ・プログラムはAからBと移植可能であると保証されない．これらのプログラム互換性に対する制限の，少なくとも主記憶装置の大きさに起因するものについては，11章に書かれている1972年中期に発表されたシステム/370仮想記憶システムではほとんど取除かれている．

上述の制限にもかかわらず，システム/360はそれまでに試みられた以上に広い性能の範囲で，実際的なプログラム互換性を与えた．OS/360と呼ばれる1つのオペレーティング・システムはすべてのシステム/360のモデル，システム・オペレーション，ユーザ・サービスのすべてのクラスに用いられるよう設計された．OS/360の開発は，主にそのような広範囲の目的とアプリケーションを持った，大きなプログラム・プロジェクトをそれ以前に経験したことがな

かったために，長期間にわたる費用のかかる過程であった．結局，個々のジョブに対し同じ論理的特徴を持つ3つのバージョンが現われた．それらは共通のジョブ制御言語(JCL)を含むが，ジョブ・スケジューリングとリソース管理機能は違っている．

本章では，われわれはOS/360のシステム性能に影響を及ぼす側面について述べる．特にスケジューリング・アルゴリズム，およびそれらの予想される空間-時間の性能の利点，欠点に重点を置く．OSの2つのマルチプログラミング・バージョン(MFTとMVT)には触れるが，OSのタイムシェアリング・オプション(TSO)は9章で述べることとし，ここでは述べない．

8-1 概略 [BR, CL, IB 1, IB 2, IB 3, IB 10, IB 11, ME, W 1 参照]

OSには3つのバージョンがある．

1. 基本制御プログラム(PCP)†
2. 固定のタスクのマルチプログラミング(MFT)‡
3. 可変数のタスクのマルチプログラミング(MUT)§

PCPは単一プログラムのオペレーティング・システムでありスプーリングさえも行なわない．これは第7.2節で議論したオペレーティング・システムに極めて似ており，そのためこれからは考慮しない．

MFTとMVTバージョンはかなりPCPと共通したところがあり，相互にも共通したところがあるが，主記憶装置とCPUの割当てのマルチプログラミング管理が異なる．これらの違いは後の節で取り上げられる．今からOS/360の全体的な論理構造とそれを述べるため用いられるいくつかの語句について主に述べる．

OS/360の選択できる3つ(PCP，MFTおよびMVT)はどれもそのジョブ制御言語(JCL)で共通のユーザ・システム間インタフェースを持っている．JCLはプログラミング言語であり，それゆえ，ユーザの多様な希望を表現することができる．JCLの複雑な文法と意味を詳しく述べることは今の議論の範囲からはずれる．しかし，JCLの3つの主な種類をそれぞれ簡単に述べ，1つの例を与えることで，いくつかの顕著な特徴の要約を述べる．

1. JOB文はいつも**ジョブ**を開始し，そしてジョブとユーザ，課金される

† PCP: primary control program
‡ MFT: multiprogramming of a fixed number of tasks
§ MVT: multiprogramming of a variable number of tasks

べき会計番号，そして時にはジョブで必要とされる主記憶装置と CPU 時間のようなパラメータやジョブ・クラスや優先度(これらについては後で論じる)のようなスケジューリング・パラメータなどを規定する.

2. EXEC 文はいつも**ジョブ・ステップ**を開始し，実行すべきプログラムを指定するか，さもなければ用意されている JCL 文の**カタログ化手続**を名指す．カタログ化手続きは JCL サブルーチンで任意の数の JCL 文を含むことができる．ユーザは各ジョブ毎に JCL シーケンスを詳細に複製する必要がないので，カタログ化手続きはユーザにとって非常に都合がよい．複製する代りに各ジョブの EXEC 文はカタログ化手続きを単に参照するだけでよく，そのカタログ化手続きはジョブの実行のために自動的に JCL シーケンスにコピーされる．JOB 文の (CPU 時間の制限のような) いくつかのパラメータはまた EXEC 文でも与えられることができ，その場合はジョブ全体にでなく，単にジョブ・ステップだけに適用される.

3. データ定義 (DD: DATA DEFINITION) 文はジョブ・ステップで用いられるデータ・オブジェクトを規定する．ステップのプログラムはしばしば数個のそのようなオブジェクトを参照するので，通常，EXEC 文に続いて数個の DD 文がある．DD 文の基本的な機能は名前の2つの型を互いに**結合する**ことである．DD カード上の1つ目の名前はプログラムの中で用いられ DD 名と呼ばれる．これはまた**ファイル**名とも呼ばれる．DD カード上の2つ目の型の名前はオペレーティング・システムに知らされ，**データ・セット**名と言われる．名前のこの2つの型はプログラマにとって捕捉しがたいが，非常に重要な便宜を与える．それらはただ DD カードのデータ・セット名を変更し，それによってプログラムのファイル名を新しいデータ・セットに結合し直すことでプログラムによって参照されるデータを変更できる．DD 文は JCL 文の中で断然複雑なものであるが，結合すべき2つの名前だけでなく，もっと多くのことを規定できる．その中にはディスクやテープのボリューム名，装置の特徴，バッファリング法などがある.

JCL が特定の場合どのように働くかを理解するために，普通の種類のジョブ，すなわち PL/1 ソース・プログラムのコンパイル，リンク・エディット，実行のための JCL カタログ化手続きを作ることを考えよう．3つの単独のプログラムが実行されるので，このジョブは3つのジョブ・ステップから成り，それらにはわれわれが勝手に選んだ名前がつけられる．それらのステップに対

するわれわれの名前は

PLIL　ソース・プログラムを OS/360 PL/1(F) コンパイラを用いてコンパイルするジョブ・ステップ．結果はオブジェクト・モジュールである．

LKED　コンパイラによって出されたオブジェクト・モジュールとコンパイラにより指定された PL/1 のサブルーチン群をリンク・エディットするジョブ・ステップ．結果はロード・モジュールである．

GO　ロード・モジュールを実行するジョブ・ステップ．データ入力はパンチされたカードから来て，出力はプリンタに送られる．

さらに進んで，われわれはすでに存在している OS システム・プログラムを用いて 2 つのジョブ・ステップ(PLIL と LKED)を実現しなければならない．OS システム・プログラムはそれらの中で決められたファイル名でデータ・セットを参照する．われわれは次にこれらのシステム・プログラムの名前と，それらが用いるそのファイル名を決めなければならない．この情報は IBM プログラマ・ガイド・マニュアルに見られる．図 8-1.1 にはこれらのプログラム名，ファイル名，そしてわれわれのジョブ・ステップで用いられる関連のあるデータ・セット名が並べられている†．われわれのジョブのためにどのようにカタログ化手続きを書くかの厳密な記述をさらに進めると，この本で問題としている以上の詳細さになる．厳密に記述しなくとも，われわれはその結果のよさがわかり，その理解ができる．図 8-1.2(a) にはわれわれの 3 つのステップのジョブの実際のカタログ化手続きが示されている．読者は図 8-2.1(b) を頼りにいろいろなファイル = データ・セットの結合を確かめることができるだろう．図 8-1.2(b) には JOB 文で始まり，そしてその後にはわれわれのカタログ化手続きの名を指し示す EXEC 文がある完全なジョブ・デックの構成を示す．残りのカードは他の JCL 文，ソース・プログラム，そしてプログラムのデータ・カードである．2 カ所使われているファイル名 SYSIN はジョブ自身のカードにつけられる．ジョブ・デックのデータ・カードにファイル SYSIN を結合するのに必要とされる DD 文は図に示されるようにジョブの中に含まれる．ファイル名 SYSIN はデータがどのステップに適用されるか指示するためにジョブ・ステップ名で(GO, SYSIN)のように修飾される．

---

† ここの例は文献 IB 11，96 頁からとられた．

ジョブ・ステップ	ファイル名	プログラム名またはデーター・セット名	目的/備考
PLL	———	IEMAA	システム・ライブラリに IEMAA という名で入っている PL/1 (F) コンパイラを実行せよ.
	SYSIN	*	コンパイラに対するソース・プログラム・カード入力.
	SYSPRINT		コンパイラからの出力リスティングとメッセージ.
	SYSLIN	&& LOADSET	コンパイラにより生成されるオブジェクト・モジュール(&& はこのデータ・セットはこのジョブのための一時的なものであることを示す).
	SYSUT 1		SYSUT 1 と SYSUT 3 はコンパイラ(IEMAA)により中間結果の記憶装置として用いられるファイルの名前である.
	SYSUT 3		
LKED	———	IEWL	システム・ライブラリに IEWL という名で入っているリンケージ・エディタ・プログラムを実行せよ.
	SYSLIN	&& LOADSET	リンケージ・エディタの入力(オブジェクト・モジュール).
	SYSLIB	SYS 1. PL 1 LIB	コンパイラにより指示されたサブルーチンのライブラリ.
	SYSPRINT		リンケージ・エディタの印字出力.
	SYSLMOD	&& GOSET(GO)	GO はリンケージ・エディタの出力 (ロード・モジュール) を受け取る一時的なデータセット && GOSET の一部である.
	SYSUT 1		リンケージ・エディタが中間結果を収めておくために用いるファイル名.
GO	———	*. LKED. SYSLMOD	現在のジョブ (*) の LKED という名のジョブ・ステップでファイル名 SYSL-MOD とされたデータ・セットにあるプログラムを実行せよ.
	SYSIN	*	データ入力はそれに続くデックである.
	SYSPRINT		プログラムの結果(出力)は印字すべきファイルである.

図 8-1.1　3ステップの OS/360 ジョブのプログラム名, ファイル名, データ・セット名の概念図

```
//PL1L       EXEC  PGM=IEMAA,
                   PARM='LOAD,NODECK'
                   REGION=52K
//SYSPRINT   DD    SYSOUT=A
//SYSLIN     DD    DSNAME=&&LOADSET,
                   DISP=(MOD,PASS),
                   UNIT=SYSSQ,
                   SPACE=(80, (250,100))
//SYSUT3     DD    UNIT=SYSSQ,
                   SPACE=(80, (250,250)),
                   SEP=SYSPRINT
//SYSUT1     DD    UNIT=SYSDA,
                   SPACE=(1024, (60,60),
                   CONTIG),
                   SEP=(SYSUT3,SYSPRINT,
                   SYSLIN)
//LKED       EXEC  PGM=IEWL,
                   PARM='XREF,LIST',
                   COND=(9,LT,PL1L),
                   REGION=96K
//SYSLIB     DD    DSNAME=SYS1.PL1LIB,
                   DISP=SHR
//SYSLMOD    DD    DSNAME=&&GOSET(GO),
                   DISP=(MOD,PASS),
                   UNIT=SYSDA,
                   SPACE=(1024,(50,20,1),
                   RLSE)
//SYSUT1     DD    UNIT=SYSDA,
                   SEP=(SYSLMOD,SYSLIB),
                   SPACE=(1024,(200,20))
//SYSPRINT   DD    SYSOUT=A
//SYSLIN     DD    DSNAME=&&LOADSET,
                   DISP=(OLD,DELETE)
//           DD    DDNAME=SYSIN
//GO         EXEC  PGM=*.LKED.SYSLMOD,
                   COND=((9,LT,LKED),
                   (9,LT,PL1L))
//SYSPRINT   DD    SYSOUT=A
```

(*a*) カタログ化手続き

```
//MYJOB  JOB '800,SMITH', REGION=100K,
//           CLASS=C, PRTY=2
//           EXEC  PL1LFCLG
//PL1L.SYSIN DD *
          (ソース・プログラム・カード)
//GO. SYSIN DD*
          (データ・カード)
//
```

(*b*) ジョブ・デック

注：1. PL1LFCLGは，カタログ化手続(a)を持っているデータ・セット名である．
2. SYSINがどのジョブ・ステップに適用されるかを示す(GO,SYSINのような)修飾されたファイル名に注意せよ．
3. データ・セット名に＊が用いられると〝//区切り記号までのレコード〟を意味する．
4. COND＝(9,LT,PL1L)は，もしステップPL1Lによって生成されるリターン・コードが9より大きければ，このステップが行なわれないことを示す．
5. SEP＝独立したデータ・チャネルでアクセスされるデータ・セットを指示する．

図 8-1.2　カタログ化手続きとそれを用いるジョブ

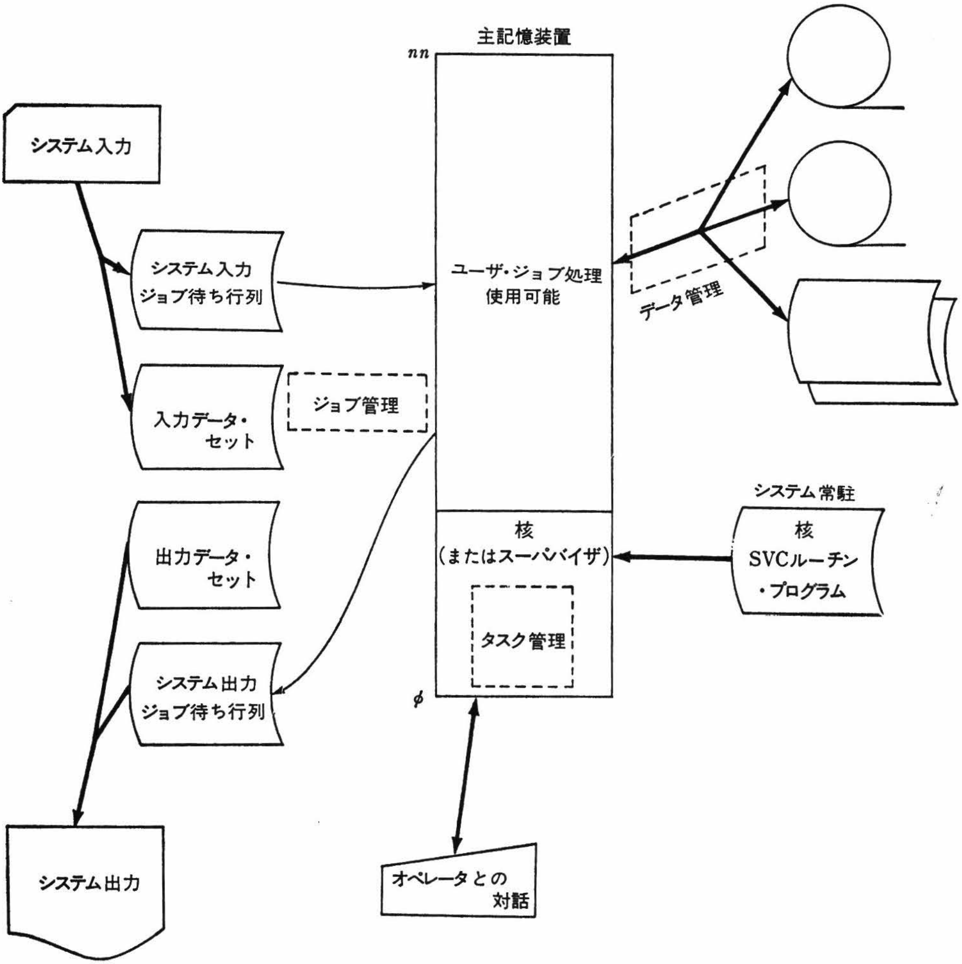


図 8-1.3　基本的な OS の機能

図 8-1.3 では簡単な概略図で OS 機能の主なクラスを示している．それらには

1. ジョブ管理
2. タスク管理
3. データ管理

が含まれる．また OS(あるいはそのことについてのいかなるシステムでも)を

理解するのに重要なのは，システム-オペレータ間の対話，システム常駐装置，およびいま簡単な例で述べたジョブ制御言語を用意することである．

ジョブ管理はジョブがシステムに（普通はカード読取機，あるいはテープ装置から）入った時からジョブが（通常，出力して）終る時まで，OS 内のジョブの動きを制御する．ジョブ管理の主な機能は

1. ジョブをシステム内に読み込み，それをジョブ入力待ち行列に入れる．
2. 各ジョブが実行時に必要とするデータ・セット，主記憶領域，I/O 装置を割当て，入力待ち行列の中のジョブをスケジューリングする．
3. ジョブの単位レコード（すなわち，プリント出力）を直接アクセス記憶装置に蓄え，後でこの出力を単位レコード装置に転送する．

である．

タスク管理はタスクを**ディスパッチ**(dispatch)するシステム・プログラミング機能である．OS/360 でのディスパッチングとは，主記憶装置中のどのジョブが次に CPU サービスを受けるかを決定することを指す．**タスク**という語は次のように OS の用語の“ジョブ・ステップ”に密接に関連している．すなわち（いつも JOB JCL 文で始まる）ジョブは（おのおの EXEC JCL 文で始まる）数個のジョブ・ステップから成る．あるジョブ・ステップが実行のためにスケジュールされ，主記憶装置に運ばれたとき，OS は**タスク制御ブロック**(**TCB**: Task Cortrol Block)と呼ばれる小さなレコードを作る．これはタスク管理によって，状態を記録し，ジョブ・ステップのプログラムをディスパッチするために用いられる．TCB が作られた時点で，ジョブ・ステップはタスクとなる．そのジョブ・ステップのためにロードされたプログラムは1つのユーザ・タスクに結びつけられる．MFT と MVT のあるオプションではユーザ・タスクはサブタスクを持つことができ，ディスパッチングのためにサブタスク自身の TCB を持つ．われわれの OS の記述では各ジョブ・ステップごとにただ1つのユーザ・タスクを考えよう．図 8-1.3 に示されるように，タスク管理機能は OS の核の中に含まれる．タスク管理はユーザ・タスクとシステム・タスクの両方のディスパッチングを制御する．

図 8-1.3 で示された OS の核はオペレーティング・システムの心臓である．核は必須のシステム機能を司どり，OS 動作の間ずっと主記憶装置中に常駐しつづける．核にはある**システム・タスク**が含まれてかり，それらはユーザ・ジョブとあるいは互いにかかわり合って，システム動作を司どる．核の他の構成要素は上で述べたタスク管理機能である．すべての I/O とチャネルの動作を

制御する入出力スーパバイザ(IOS)もまた核に含まれる.

核より頻繁に呼ばれないシステム・プログラムを主記憶装置中に常駐されるかどうかは，そのコンピュータ運用での自由裁量である．スーパーバイザ・コール(SVC)プログラムは1つの例である．ユーザ・プログラムが核からのサービスを要求するとき，たとえばSTART-I/Oなどが起動される．もしSVCに応答するシステム・プログラムが主記憶中に常駐していれば，すぐにそれに制御が渡される．しかし，もし常駐していなければ，ユーザの要求をサービスする前に，核はSVC応答プログラムをディスクから(核の中の領域である)**一時領域**に転送しなければならない．(純粋に速度の点から)ディスクへのアクセスは遅いので，しばしば参照されるSVCの応答プログラムを核の中に常駐させると有利である．しかし，その犠牲として核の大きさが増え，ユーザ・プログラムのために残された主記憶装置が減る．主記憶装置中に常駐するかどうか選択できるシステム・プログラムの，他の例は後で議論しよう．どのシステム・プログラムとテーブルを主記憶中に恒久的に常駐させるかを決めることはOS/360では非常に重要なユーザ側の責任であり，システム性能に非常に大きく影響しうる．これについては後でもっと議論する．

データ管理はOSによって遂行される主なシステム機能の3番目のものである．データ管理はすべてのI/O動作を制御し，あるコンピュータ設備がシステム・データ・セットとユーザ・データ・セットの両方を作り出し保持し，そしてアクセスすることができるようなサービスを与える．システム・データ・セットはライブラリであり，SVCのようなシステム・プログラムと実行形式(ロード・モジュール)，あるいはソース・プログラム形式で入れられているユーザ・プログラムを含む．これらのライブラリは一般に(ディスクのような)直接アクセス記憶装置に収められている．ユーザ・データ・セットは1つ以上のジョブに特有の，給料マスタ・ファイルのようなレコードを入れるのに普通使われる．そのようなデータ・セットもまた直接アクセス装置，あるいはテープ装置に入れられる．データ管理によって与えられる主なサービスは，どのようにOSがマルチプログラミングを実現するかを記述するのに必須ではないので，この話題はこの章ではこれ以上議論しない．

OSの**システム常駐部**は直接アクセス記憶装置中の1つの領域(あるいは複数の領域)であり，システム・ライブラリに置かれているすべてのシステム・プログラムのコピーをユーザ・プログラムと同様に入れておく．図8-1.3ではシステム常駐装置に収められているシステム・プログラムの例として核とSVCルーチンが見られる．システム常駐領域は3つの主な機能を果たす.

1. IPL プログラムがシステム常駐部から主記憶装置中に核のコピーをロードする時にシステム動作が開始される．それから制御は核に渡される．
2. (多くの SVC プログラムのような) 一時的なシステム・プログラムはずっと主記憶中に常駐する必要はなく，コピーがシステム常駐部から主記憶装置に必要に応じて核の制御の下に転送される．
3. ユーザ・プログラムはシステム常駐パックのライブラリに収められ，実行時に主記憶装置に転送されうる．こうして頻繁に使われるプログラムをパンチカードのデックからロードする必要がなくなる．

オペレータと呼ばれる人は OS を走らすことを要求される．彼の仕事には

1. IPL 手続きでシステムを起動する．
2. パンチ・カードのデック，あるいはテープの形で入力ジョブを入れ，(プリントされたリスティングのような)最終出力をユーザの所へ発送する．
3. リーダ・インタプリンタ(これはカード読取機やテープ装置からシステムにジョブを読込む)やイニシエイタ・ターミネイタ(これは処理すべきジョブを選択し，そのプログラムを主記憶装置に転送する) のようなあるシステムプログラムを起動したり，停止させたりする．
4. ジョブ処理中に(コンソール・メッセージの指示に従い)要求されたテープ・リールやディスク・パックを装塡する．
5. システムの故障の際に回復手続を試みる．

ことが含まれる．

OS を操作するには，図 8-1.3 に示されるように，オペレータと OS 自身の間にかなりの量の対話が必要である． この**オペレーター-システム対話**は**マスタ・スケジューラ**と呼ばれる 1 つの OS プログラムによって制御されている．マスタ・スケジューラは OS 動作中，常に主記憶の核の中に常駐している．それはオペレータにすべての**システム・メッセージ**を発する．それらのメッセージはシステム・コンソールと呼ばれるタイプライタや CRT 表示装置に出力される．マスタ・スケジューラはまたオペレータがシステム・コンソールから入れたすべての**コマンド**を解釈し実行するプログラムをも含んでいる．

単一ジョブで述べられたスケジューリング機能は**図 8-1.4** を参照しながら簡単に要約される．図ではジョブがシステムで処理される時のジョブの状態を示している．各状態はまたその時ジョブやタスクを保持している記憶装置のタイプによっても区別される．状態間の線は状態遷移を示し，その状態遷移を引き起こすシステム・プログラムの名前がつけられている．

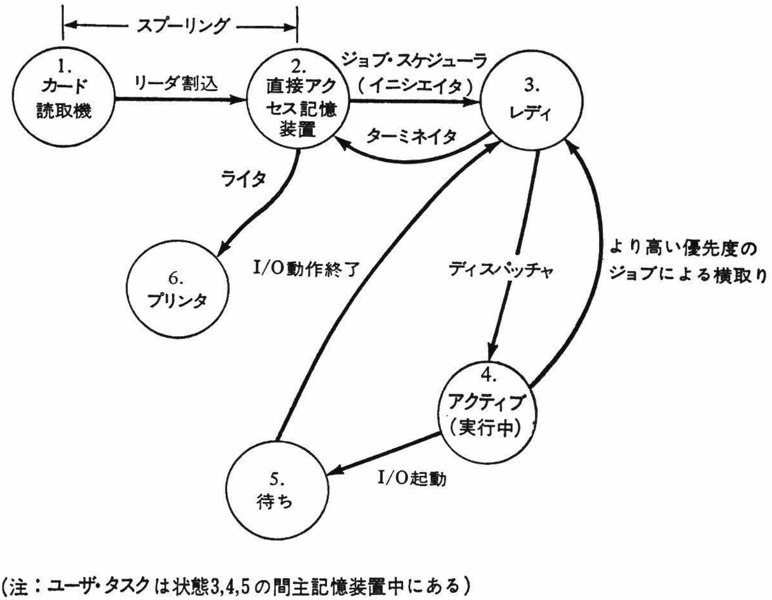


図 8-1.4　状態遷移を引き起こすシステム・プログラムを示した OS ジョブの状態図

この簡単な概略図の目的は OS/360 の主な機能のいくつかを略記することである．MFT，MVT バージョンは共にこれらと同じ主機能を持っている．MFT と MVT の違いは，どのようにマルチプログラミングを管理するかのもっと詳細な記述でのみ理解される．

8-2　OS/MFT [IB 4 参照]

図 8-2.1 は MFT システムでどのように主記憶装置が構成されているかを示している．主記憶装置の残りは，**システム・パーティション**と**問題プログラム・パーティション**(すなわちユーザ・パーティション)の 2 つのタイプのパーティションに分けられている．システム・パーティションはカード入力をディスクにスプールするリーダや，出力をプリンタ，テープ，カード・パンチなどにスプールするライタのようなシステム・プログラムのために用いられる．導入設置時には(システム・パーティションと問題プログラム・パーティションを含めて) 52 までのパーティションを指定できるが，1 から 5 のシステム・パーティションが多分 MFT 装置では代表的なものであろう．

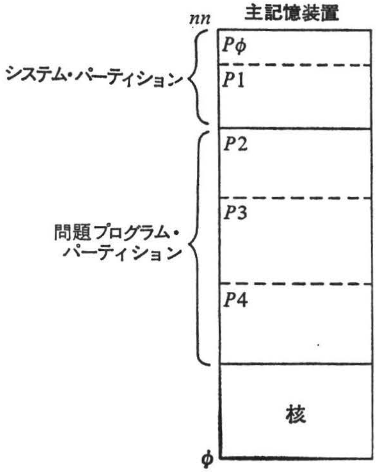


図 8-2.1 MFT 主記憶構造

ジョグ・スケジューリングはジョブを問題プログラム・パーティション(PP)に割付けることから始まる．ジョブが一旦 PP パーティションに割付けられたら，そのジョブ・ステップのすべてがそのパーティションの中で実行される．MFT システムでは 15 以上の PP パーティションは指定されないので，15 がマルチプログラムされるユーザ・ジョブの数の最大値である．この制限はシステム/360 とシステム/370 の第 7 章で述べた記憶保護方式を MFT が用いていることによる．その 4 ビットのキー・ロックは 16 個の異なったキー値を与える．ゼロのキーはすべてのシステム・プログラム，すなわち，核とシステム・パーティションで実行されるすべてのプログラムのために用いられる．残りのキーが PP パーティションに割付けられうる．そのキーでシステム・プログラムはそれらの PP パーティションの境界内の主記憶装置を参照できる．

各 OS システムは**システム・ジェネレーション**と呼ばれる処理で生まれ，その生活を始める．ここで，オペレーション・システムの原コピーにオプションが与えられ，特定のコンピュータ設備のリソースとオプションに適したオペレーティング・システムのバージョンが生成される．

MFT 装置はシステム・ジェネレーション時にシステム・パーティションと PP パーティションの両方の数を指定する．パーティションの数は新たなシステム・ジェネレーション(これは長い処理である)でしか増すことはできない．

選択されたパーティションの数に関するもう1つの重要な点は，各パーティションがコントロール・ブロックとしていくらかの記憶空間を核の中に要求することである．各パーティションの大きさも，システム・ジェネレーションの最中に2kの倍数で定義される．しかし，パーティションの大きさは動作中にオペレータにより手で**再定義**できる．たとえば，2つの隣接するパーティションが1つの大きなパーティションとなるように結合されることができる．

パーティションは名前 $P0, P1, \cdots, Pn$ で区別される．その名前の整数はそのパーティションを占めるジョブのタスク・ディスパッチング優先度を決める(これについては後で述べる)．差し当り，パーティションのディスパッチング優先度は主記憶装置の先頭から下へラベルの順につけられ，パーティションに付けられた数字が小さいほど高いディスパッチング優先度を持つとする．

われわれは**ジョブ・スケジューリング**は単にユーザ・ジョブをPPパーティションに割付けることであると前に述べた．MFTジョブ・スケジューリングは前に一般的に述べたジョブ管理機能の一部である．**図8-2.2**は用いられるシステム・プログラムを示すもっと詳細なMFTのジョブ管理機能の図である．その図は図8-1.3の拡張版である．丸で囲まれた数字はこれらの機能を通るユーザ・ジョブの流れを示している．ユーザ・ジョブはパンチ・カード，あるいは磁気テープからMFTに入る．各ジョブにはジョブ，ジョブ・ステップ，要求するデータ・セットと装置を識別するJCL文を含んでいる．入力ジョブの流れは**システム入力**とよばれる．MFTはシステム入力をSPOOLする．リーダ-インタプリタのシステム・プログラム(リーダと呼ばれる)は以下の機能を果たしながらシステムにジョブ・ストリームを転送する．

1. ジョブのJCLで呼ばれたJCLカタログ化手続きをジョブ・ストリームに挿入する．
2. JCL文を**解釈**し，MFTがジョブをスケジュールするために必要とする制御ブロックを作り出す．もしジョブ内にJCLエラーがあれば，(適当なエラー・メッセージと共に)そのジョブは無効にされる．
3. ジョブとそれに関する制御ブロックを，直接アクセス記憶装置に収められている**システム(入力)ジョブ待ち行列**に置く．後でジョブはそこからスケジュールされる．
4. ジョブの入力データにラベルを付け，直接アクセス記憶装置に収める．ジョブ実行中，その入力は直接アクセス記憶装置から読まれる．

MFTのシステム入力はスプールされているので，リーダ・プログラムはユ

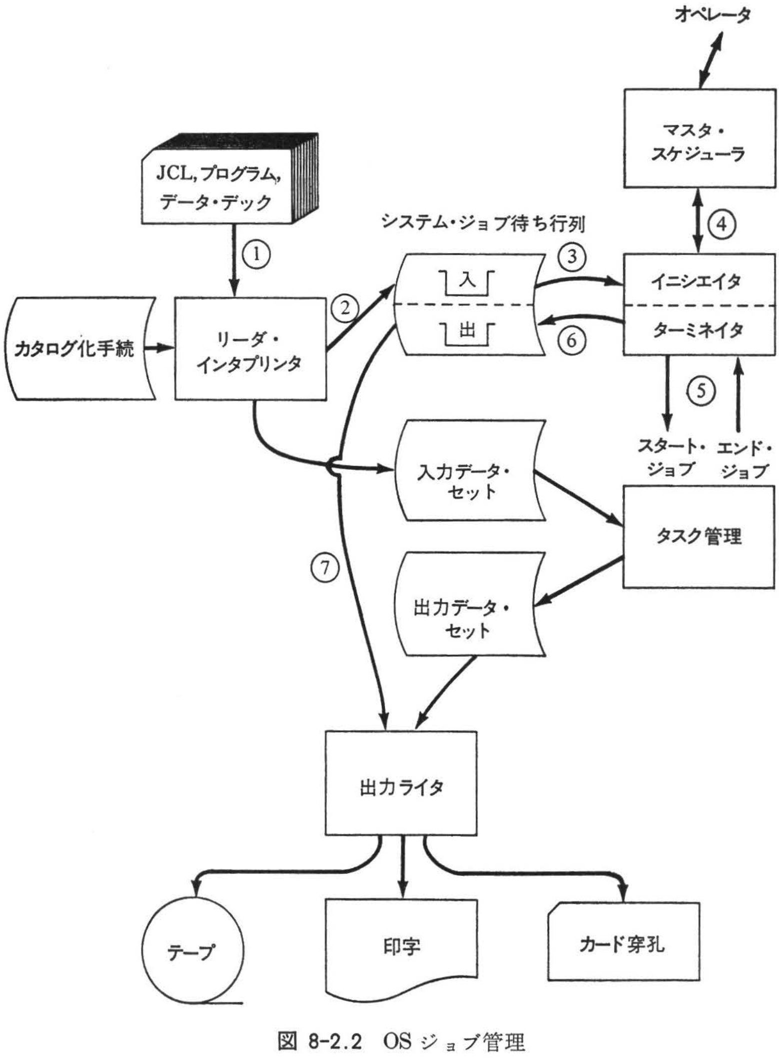


図 8-2.2 OS ジョブ管理

ーザ・ジョブとマルチプログラムされる．言い換えれば，リーダがジョブの準備をしている間に，他のジョブは主記憶装置中にありCPU実行やI/O動作などのいろいろな状態にある．

一旦ジョブがシステム・ジョブ待ち行列に格納されたら，それらは実行をスケジュールされうる．図8-2.2で見られるように，ジョブ・スケジューリング

は**イニシエイタ・ターミネイタ**・システム・プログラムによって行なわれる．PP パーティションを起動するために，オペレータは(マスタ・スケジューラ・プログラムを通して)パーティション内のイニシエイタを開始させるコマンドを発する．イニシエイタはそれで，一般にそのパーティションのみの，ジョブのスケジュールを始めることができる．こうして，オペレータはシステム内の動作中のパーィィションの数を制御できる．起動されたイニシエイタを持つ各 PP パーティションは自分自身のスケジューラを持つ論理的なマシンとして考えられる．

稼動している間，ジョブ・スケジューリングはあるジョブが PP パーティションで終了した時に始まる．まず今終ったジョブのプログラムをオーバレイして，イニシエイタ・プログラムのコピーがロードされる．イニシエイタのターミネイタ機能によって，終了したジョブは**システム(出力)ジョブ待ち行列**に置かれる．そこから，その**システム出力**はプリンタ，カード，あるいはテープ装置に対してスケジュールされる．ターミネイタはまた終了したジョブに割付けられていた I/O 装置を解放し，それらを MFT の制御に戻す．そしてイニシエイタは PP パーティションに新しいジョブ(あるいは，ジョブ・ステップ)をスケジュールする．MFT ジョブ・スケジューリング・アルゴリズムを用いて(これについては後で詳しく説明する)，イニシエイタはシステム・入力ジョブ待ち行列から“最善”の候補を選ぶ．そしてイニシエイタは(ジョブの JCL 文で)要求されたデータ・セットと I/O 装置を割当てなければならない．しばしばユーザは特定の装置を示さず，たとえばディスクとかテープとか，単に装置の種類だけを要求する．(オペレータに特定の“ボリューム”，すなわち，テープ・リールまたはディスク・パックを装塡するようにとのメッセージは JCL 情報から OS によって発せられる)．このように非常に後の方でジョブは特定の I/O 装置に結びつけられ，それゆえ実際に使用可能なリソースへのスケジューリングの適用性が増加する．すべての I/O 装置が割当てられた後，イニシエイタは MFT の核にある再配置ローダ・プログラムに制御を渡すことにより，ジョブの最初のジョブ・ステップを始める．他の OS/360 バージョンと同様に MFT はロード時(静的)再配置を行なう．そのローダはジョブ・ステップのロード・モジュールのアドレスを処理し，その PP パーティションの始点からのアドレスとする．プログラムはそれからそのパーティションに転送される．れそから制御がそのプログラムに渡され，ユーザ・タスクは実行を開始する．この時点から，ユーザ・タスクの CPU へのアクセスは，**図 8-2.2** に示されるように，MFT のタスク管理機能の制御下にある．ジョブ終了時に，前に

述べたように，イニシエイタ・ターミネイトはPPパーティションへロードされ，ターミネイタはジョブを終了させ，イニシエイタは入力ジョブ待ち行列から他のジョブをスケジュールする．MFTは再配置ローダを持っているので，プログラム・ロード時までプログラムは主記憶装置の位置に結び付けられず，ジョブは（プログラムやJCLを変更することなく）走るごとに異なったパーティションにスケジュールされることができる．しかし，動的再配置ができないので，一旦プログラムが実行を開始すると，主記憶装置の同じ領域で終了しなければならない．そしてMFTでは，一旦ジョブがあるPPパーティションにスケジュールされると，そのジョブ・ステップのすべてはそのパーティションで実行しなければならない．

さてMFTのジョブ管理を振り返って見て，次にどのようにMFTがシステム・ジョブ待ち行列からジョブをスケジュールするか調べる．どのようなスケジュール規則をMFTイニシエイタが用いて，待ち行列から次のジョブを選ぶのであろうか．図8-2.3にはこれがどのように行なわれるかを知る手助けとなる例を示す．*P*0と*P*1はライタ，リーダ・インタプリンタを持つシステム・パーティションである．また*P*2から*P*4の3つのPPパーティションがある．*P*2と*P*4はたとえばJOB7とJOB8にスケジュールされており，*P*3は他のジョブのための準備ができていると仮定しよう．イニシエイタのコピーはその時*P*3の中にある．ジョブ・スケジューリングは**ジョブ・クラス別**スケジューリング方式で行なわれる．文字*A*から*O*で表わされるジョブ・クラスはPPパーティション(SYSGEN時またはオペレータによって)とユーザ・ジョブ(ユーザによって)の両方に割付けられている．すでにパーティションに割付けられたクラス値が付けられたジョブ待ち行列の中のすべてのジョブのうちで“最良の”候補をイニシエイタはスケジュールする．さてこのスケジューリングの過程を詳しく調べよう．

ジョブのクラスは図8-2.3のJOBカードの絵に見られるようにJOB JCLカードにCLASS＝パラメータとしてユーザによりパンチされる．ジョブに対するジョブ・クラスの割付けは図に示されるように入力待ち行列を各クラスに1つずつの幾つかの部分待ち行列に分割することと考えられる．数個のジョブが同じクラスを持つかも知れないので，そのタイブレーキング方法が必要となる．最初のタイブレーキング方法はユーザがやはりJOBカードに0（最も低い優先度）から13の間の**優先度**値(PRTY＝)を与えることができるようになっている．もしPRTY値を用いてもなおイニシエイタが唯一のジョブを持てなければ，残りのタイブレーキングはジョブが各待ち行列に入った順番(すな

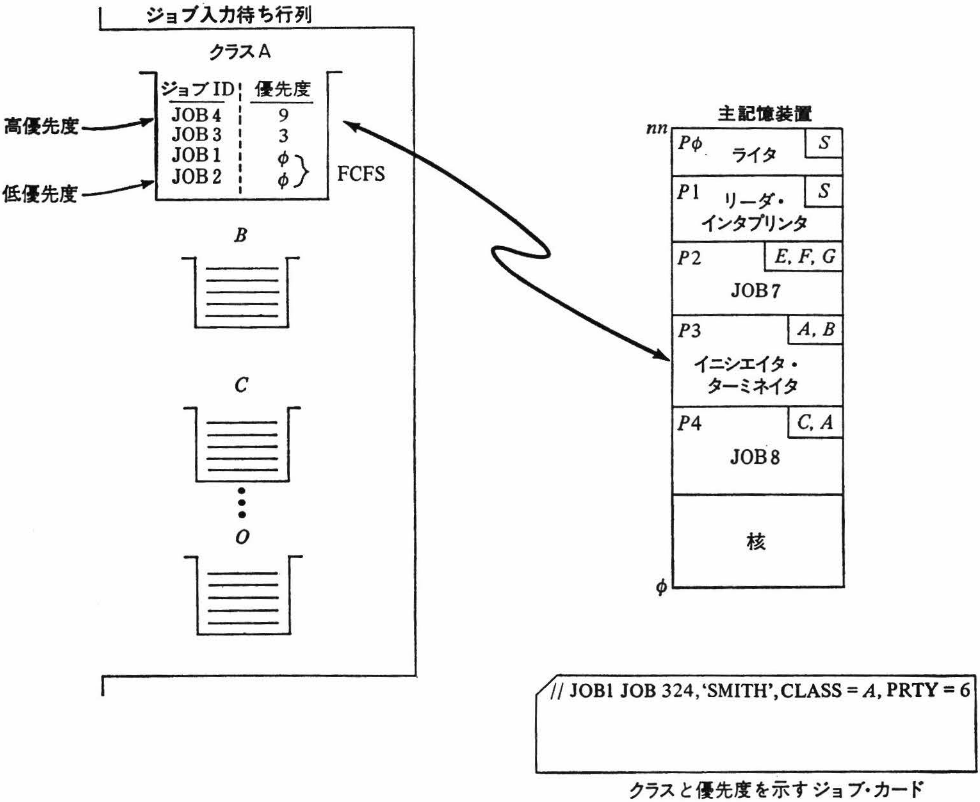


図 8-2.3　ジョブ・クラスと優先度によるスケジューリング

わち優先度とクラスで FCFS) で行なわれる.

考えられる 15 のジョブ・クラス A-O の中の**1つから3つ**が各 PP パーティションにオペレータにたって割付けられる. あるパーティションのジョブはそのパーティションに割付けられたクラスに対応するクラスの待ち行列からのみスケジュールされる. あるパーティションに割付けられた最初のジョブ・クラスはそのパーティションでのジョブ・スケジューリングの最高の優先度が与えられる. もし2番目があれば, それには2番目の優先度, 3番目があれば最低の優先度が与えられる. 図 8-2.3 はジョブ待ち行列とパーティションに割付けられたクラスでのあるスケジューリング問題を示す。パーティションのクラスは各パーティションの右上端に, クラス優先度は左から右に示される. われわ

れはシステム・パーティションにクラス S を割付けた．この例から，MFT における ジョブ・スケジューリングについていくつかの性質がみられる．$P3$ のイニシエイタはジョブ・クラス待ち行列 A あるいはジョブ・クラス待ち行列 B からジョブをスケジュールするであろう．$P3$ に割付けられたクラス・リストの最初のものが A であり，それゆえ A は B より $P3$ に対してより高い優先度を持つので，待ち行列 A のすべてのジョブが最初にスケジュールされる．待ち行列 A が数個のジョブを持っているとすると，イニシエイタは待ち行列 A から“最良の”候補をスケジュールするであろう．待ち行列 E, F, G の中のジョブは $P2$ の中でしかスケジュールされない．また図8-2.3では同じジョブ・クラスが異なるパーティションに割付けられている点に注意されたい．たとえば，$P3$ と $P4$ は共にクラス A のジョブをスケジュールするが，$P3$ ではクラス A のジョブは高いスケジューリング優先度である．例では6つだけのジョブ・クラスが PP パーティションに割付けられているが，MFT は15のョジブ・クラス待ち行列を持ちうる．たとえばクラス G に割付けられたジョブがシステムに入ると，それはクラス G ジョブ待ち行列に置かれる．このジョブはオペレータがクラス G を PP パーティションの1つに割付けるまでスケジュールされることができない．丁度オペレータがシステム動作中にパーティションの大きさを再定義できるように，オペレータはまたパーティションへのクラスの割付けも再定義できる．

ユーザはジョブにクラスを割付ける．システム・オペレータがあるいはシステム導入時にパーティションにクラスを割付ける．それゆえ，クラスに関する操作の意味は少なくとも一部はユーザによって決められる．システム導入時にはユーザはどのようにジョブへクラスのラベルを割当てるかを示唆される．たとえば，ある導入時には特定のクラス，あるいはクラスの集合が I/O 向きのジョブに割付けられるように指定するかも知れない．同様に，あるクラスの集合が CPU 向きのジョブのために確保されているであろう．ジョブ・クラスはまたジョブの大きさと関係がある．たとえば，80k から100k の大きさを要求するすべてのジョブは，あるクラスあるいはクラスの集合を用いて100k の PP パーティションに割付けられる．

MFT パーティションの大きさは**静的に**割当てられ，すなわち，オペレータは含まれているパーティションが動作していない時に，パーティションの大きさを定義するので，MFT のユーザは(ジョブにクラス値を割付けるときに)そのクラスがそのジョブを走らせるのに充分な大きさのパーティションで扱われることを保証するように注意しなければならない．一旦ジョブがパーティショ

ンに割付けられるとそのジョブ・ステップのすべてはそのパーティションで実行しなければならない．それでジョブの大きさ(主記憶装置要求量)はジョブの中の最大のジョブ・ステップの大きさと考えられる．もし，あるジョブが大きさが 70k，90k，80k の3つのステップを持つなら，そのジョブは 90k のパーティションの中にスケジュールされなければならない．そのジョブが，もしたとえば 200k のパーティションの中にスケジュールされたら，そのジョブの3つのジョブ・ステップ実行中におのおの 130k，110k，120k が無駄となる．たとえ 90k のパーティションとしても最初のステップで 20k，3番目のステップで 10k の使わない空間がでる．この無駄な主記憶装置は内部**細分化**の例であり，前に静的再配置システムに関して一般的に述べられた問題である．MFT における細分化の程度はどのように巧妙にそのコンピュータ設備のオペレータがパーティションを形成するか，また，どのようにうまくユーザがジョブの大きさをパーティションに割付けるかによる．この“**適合化**”問題は I/O による制限のあるジョブには高優先度のパーティションを，CPU による制限のあるジョブには低優先度を割付けるという目標と天秤にかけ，システム・リソースの全体的な高い使用率を達成しなければならない．後者についてはもう少し後で議論しよう．クラス・スケジューリングには MFT ユーザと装置がジョブ・クラスに意味を与えることができるという利点があるが，システムでなく，それらの人々にすべてのシステム・リソースの高使用率の責任を課すという欠点がある．

MFT においてあるジョブに割付けられたクラスと優先度は，ジョブが受ける**ディスパッチング優先度**に直接的にはなんら影響しない．クラスと優先度はジョブ・スケジューリングのパラメータとしてのみ用いられる．しかしながら，ジョブ・クラスはパーティションを決め，その中でジョブがスケジュールされるので，間接的にディスパッチング優先度に影響する．前に述べたように，これはパーティションの数字ラベルがディスパッチング優先度を決めるからである．この様子は後の MFT タスタ管理の記述の中で示される．

MFT のジョブ管理機能を要約するために，われわれはあるジョブのシステム内の流れをたどることにする．例として，PL/1 のコンパイル(1ステップのジョブ)を用いる．**図 8-2.4** にそのジョブのシステム入力を示す．ジョブ・カードはジョブ，そのスケジューリング・クラス(F)，スケジューリング優先度(12)を見分ける．実行(EXEC)カードはジョブ・ステップを区別する．(この場合は PL1FC と呼ばれる JCL カタログ化手続で間接的に名前がつけられる.) 実際には，これは PL/1 コンパイラのコピーがジョブ・スケジューリ

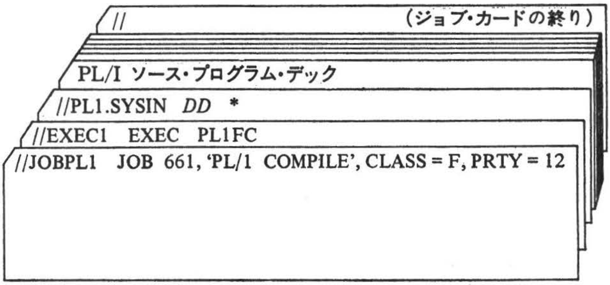


図 8-2.4 ジョブ・ストリーム例

ング時，システム・ライブラリから割付けられたPPパーティションにロードされることを指定している．**データ定義カード**(DD)はジョブ・ステップ，この場合にはPL/1 Fコンパイラ，によって要求される入出力データ・セットを示す．これらのデータ・セットは(示されてはいないが)PL1FCと名付けられた手続き中のDD文によって，たいてい規定される．DD * 文は＊によって後に続くPL/1ソース・プログラム・カードがジョブ・ステップ(PL/1 Fコンパイラ)のためのデータ・セットであることを規定する．ソース・プログラムの後に//区切りカードが続き，このジョブのシステム入力の最後を示す．

図 8-2.5 はMFT内のこのPL/1ジョブの流れを図式的に示す．そのジョブをクラス *F* に置き，ソース・プログラム入力データを直接アクセス記憶装置にスプールする常駐リーダを仮定する．*P*4 だけがクラス *F* のPPパーティションであり，それゆえ，われわれのPL/1ジョブは *P*4 の中にスケジュールされなければならない．このジョブが開始された後では，実行中にソース・プログラム入力を直接アクセス装置から読む．また実行中にすべての印字出力は直接アクセス記憶装置の出力データ・セットに置かれる．ジョブが実行を終了したとき，イニシエイタ・ターミネイタが呼ばれ，結果をシステム出力待ち行列に置く．FCFS規則を用いて，システム・ライタ・プログラムはPL/1ジョブの出力を主記憶中のスプール・バッファ領域を介して，直接アクセス記憶装置からシステム・プリンタにスプールする．図8-2.5の主記憶装置マップから別の考察をすることができる．核の大きさは70kバイトである．MFTの核の大きさの最小は約38kであるが，複数個のパーティションや常駐SVCプログラムのようなオプションの選択が(ある利益をもたらすが)核の大きさを広げる．MFT機能，その結果の利益，そしてそれらの要求するリソースをうまく釣合せることがMFTシステムの“調整”時の基本的な目的である．また

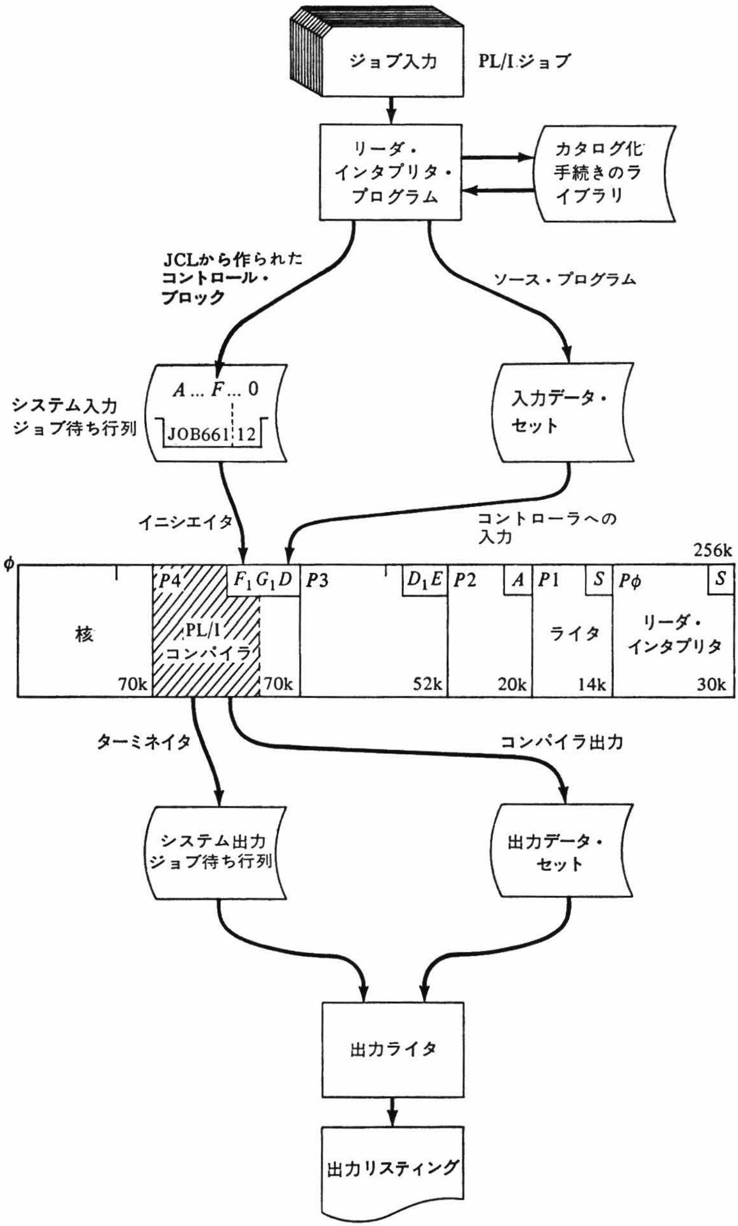
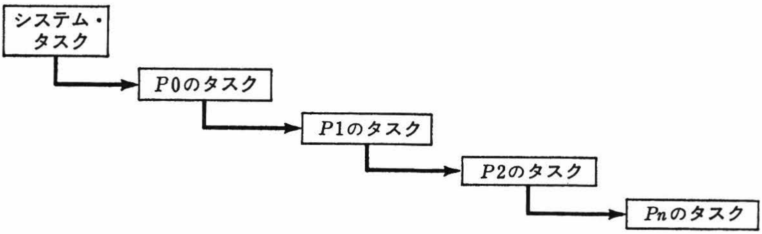


図 8-2.5　MFT におけるジョブの流れ

図8-2.5ではリーダ，ライタ機能は主記憶装置に常駐であることに注意しよう．小さな主記憶装置のMFTシステムでは，オペレータはこれらのスプーリング機能とユーザ・パーティションの間で主記憶装置を“スケジュール”しなければならない．図8-2.5のパーティション $P1$ は大きさがたった20kである．MFTのイニシエイタの最小の大きさは約30kであるので，$P1$は“従属”パーティションである．それで $P1$ は他のパーティションの中のイニシエイタによってスケジュールされなければならない．これは結果として $P1$ のスケジューリングが遅れることになるが，もしそのコンピュータ設備が多くの小さなジョブ(20k以下の)を持っていれば，主記憶装置の内部細分化を避けることができる．

MFTにおけるジョブ・スケジューリングはどのジョブ(ユーザ・タスク)がPPパーティションに転送されて，I/Oリソースが割当てられるか，を決める．MFT核のI/Oスーパバイザ(MFTデータ管理機能の一部)はI/O動作を，したがってシステムのチャネルを**到着順**(FCFS)にスケジュールする．**タスク・ディスパッチャ**(MFTのタスク管理機能の一部)はMFTの核の中や，主記憶装置のパーティションの中にあって動作中のタスクの間でCPUをスケジュールする．MFTの残りの議論の中で，この**タスク・ディスパッチング**(または単にディスパッチングと呼ばれる)について述べる．

MFTは第7章で述べた手法に似た［**横取り**(preemption)のある］**静的優先度順**(HSFS: highest-static-priority-first-service)ディスパッチング手法を用いる．**システム・タスク**はMFTの核とシステム・パーティションに駐在する．マスタ・スケジューラのような核の中にあるシステム・タスクは，システム生成時に最も高い優先度が割付けられる．**ユーザ・タスク**はPPパーティションから実行される．MFTの設計はディスパッテング優先度を $P0$ が最高で，次は $P1$ というようにパーティションのラベルに固定している．MFTユーザはこのパーティション優先度を変更することはできない．前に述べたように，ユーザは実際のディスパッテング優先度に間接的にしか影響を与えられない．これはユーザの基本的な制約として，そのジョブを走らせるのに充分な大きさがあるパーティションに割付けられたジョブ・クラスをジョブに割付けなければならないからである．このクラスを割付けられたパーティションが1つだけである場合には，彼のジョブはそのパーティションの中で実行しなければならず，そのジョブのディスパッチング優先度は完全にパーティション番号で決められてしまう．しかし，もしジョブが2つ以上のパーティションで処理されることができるクラスを割付けられたら，ジョブはそれが走るパーティショ



n→ MFTで生成される最も低優先度の(タスク)パーティション

図 8-2.6　MFT ディスパッチング待ち行列

ンによって異なったディスパッチング優先度にでくわすであろう．

図 8-2.6 は MFT の核の中のタスク・ディスパッチャによって用いられる**ディスパッチング待ち行列**を図式的に表わしたものである．優先度のシーケンスは待ち行列のレベルで示される．待ち行列の要素は**タスク制御ブロック**と呼ばれる．MFT はまず準備できているシステム・タスクのどれかをディスパッチする．もし，どのシステム・タスクも CPU の実行が必要でなければ，$P0$ にあるシステム・タスクであろうとユーザ・タスクであろうと，それが実行の準備ができていると仮定すると，$P0$ の中のタスクがディスパッチされる．2 つの事象のうちの 1 つが起こるまで $P0$ のタスクは実行し続ける．2 つの事象とは，

1. たとえば，I/O 動作のため待ち状態に入る．
2. より高い優先度を持つ，すなわち，より小さなパーティション番号のタスクがたとえば I/O 動作の終了によって(割込みで知らされ)CPU を要求する．

である．ディスパッチング待ち行列の要素の最大数は，核の中のタスクの数とシステムのパーティションの数によって決まる．

MFT でのジョブ・スケジューリングは HSFS ディスパッチング規則のためにシステムの CPU と I/O リソースの使用が“まずく”も“良く”もなりうる．C-バウンド・ジョブ† が，最高の優先度を持つ PP パーティションにスケジュールされたときは，結果としてまずいディスパッチッグとなる．C-バウンド・ジョブは待ち状態に滅多に入らず，より低い優先度のパーティションにある I/O-バウンド・ジョブ† は実行される機会が少ない．ジョブの似た組合

† C バウンド・ジョブ，I/O バウンド・ジョブ：§6.4 参照

せでも，I/O-バウンド・ジョブを高い優先度のパーティションにスケジュールされると，“良い”ディスパッチングが得られる．それらのジョブは(I/O 処理の間)しばしば待ち状態に入り，低い優先度の C-バウンド・ジョブを実行する時間がある．前の“まずい”ジョブ・スケジューリング管理よりも大きなシステム稼動率，したがって高いスループットが得られるであろう．

ジョブ・スケジューリングによって，MFT ユーザはバッチ向きのマルチプログラミング環境下において，2つの主要な目標を達成しようとする．すなわち，(1) 良い CPU ディスパッチング，したがってシステム・スループットの向上，(2) 主記憶装置の細分化を少なくし主記憶装置リソースの有効利用，である．

静的に割当てられたパーティションの大きさと MFT でのジョブ・ディスパッチングの優先度の単に間接的な制御では，両方の目標を達成することは困難である．MFT からの手助けはほとんどなく，これらの目標を達成する責任のほとんどはユーザと導入・設置の肩にかかっている．OS/MVT システムはその主記憶装置割当て手法とディスパッチング優先度割付けでジョブ・スケジューリングを容易なものとしている．

8-3 OS/MVT [IB 5, IB 6 参照]

OS/MVT は多くの点で OS の MFT バージョンと似ており，前の MFT の記述の多くは MVT にもまた当てはまる．この節では OS/MVT の MFT に対する主な2つの機能的相異点を述べよう．(1) 主記憶装置管理手法，(2) タスク管理機能．

各ジョブ・ステップはその主記憶装置要求量 (*nnn* k バイト) を示すユーザが指定する JCL パラメータ(REGION = *nnn*)を持って MVT に入る．ジョブ・ステップがスケジュールされるとき，要求された量の主記憶装置が割付けられる．前もって指定された大きさのパーティションにあるジョブ全体を割付けなければならない MFT より効率的になるだろう．このように柔軟性において第一の大きな改善がみられる．すなわち，MVT は MFT と異なり1つのジョブの各ジョブ・ステップごとの異なった主記憶装置要求量に対処できることである．

図 8-3.1 は OS/MVT における主記憶装置の簡単な配置である．MVT の核は主記憶装置の下端にロードされる．機能の点では，その核は MFT の核に似ており，タスク管理と I/O スーパバイザを含む．主記憶装置の上端には，MVT **リンク・パック領域**(LPA: *link pack area*)がある．LPA 内の基本的

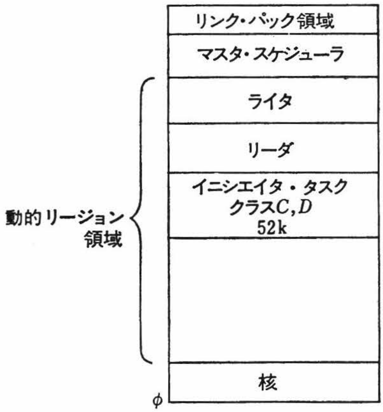


図 8-3.1　イニシエイタ起動後の MVT での主記憶装置

なルーチンは MVT イニシエイタ・ターミネイタ(イニシエイタ)プログラムと共に，主に主記憶装置管理とジョブ・スケジューリングのために用いられる．これらのルーチンは MVT の動作中はずっと主記憶装置中に常駐している．これらのルーチンだけで，LPA の大きさは約 6k バイトである．しかし，MVT 装置は LPA にリエントラント・システム，ユーザのルーチン，システム・ライブラリのディレクトリをロードするかも知れない(また普通そうしている)．もしそれらが LPA になければ，それらをフェッチするのにディスクのアクセスが必要となるので，それらが LPA にあることはシステム性能を非常に改善することができる．たとえば，頻繁に用いられるアクセス法は LPA の中に常駐させられるであろう．常駐しているアクセス法を要求するユーザ・ジョブは LPA 中のコピーに単に "リンク" される．多数のユーザ・ジョブは同時にアクセス法の同じコピーを共有するので，主記憶装置の空間を節約できる．頻繁に参照される SVC もまた LPA に入れておける．また，直接アクセス記憶装置上の SVC ライブラリ(それに他のシステム・ライブラリ)へのディレクトリも LPA 中に常駐でき，これはディスクのアクセス時間の量を非常に減らす．LPA 中に常駐させるもののリストは IPL 時に MVT オペレータによって指示される．それは新たな IPL によってのみ変更される．

MVT マスタ・スケジューラは MFT のものと機能的には似ている．それはオペレータと MVT システムの主要なコミュニケーション・リンクである．もしオペレータがイニシエイタを起動するときは，そうせよとの彼のコマンド

はマスタ・スケジューラの**コミュニケーション・タスク**によって読まれ，解釈され，実行される．同様に，オペレータはマスタ・スケジューラを通してリーダとライタを起動できる．

核，リンク・パック領域，マスタ・スケジューラによって使われる主記憶装置は，その内容が動作中ずっと主記憶常駐であるので，**固定領域**と呼ばれる．主記憶装置の残りの部分は，図8-3.1に示されるように，**動的リージョン領域**と呼ばれる．MVTにおいては，**リージョン**(region)とは主記憶装置の連続したブロックであり，このリージョンは動的リージョン領域からシステム・タスクやユーザ・ジョブ・ステップに2kの倍数で割付けられる．MFTのパーティションとは異なり，リージョンの大きさはシステム・タスクやユーザ・ジョブ・ステップの主記憶装置要求に基いて動的に割当てられる．リージョンは動的リージョン領域で上から下に割付けられる．

動的リージョン領域に入るシステム・タスクは主としてリーダ，ライタとイニシエイタである．各システム・タスクはそれ自身のリージョンに入る．MVTのリーダ・インタプリタとライタはMFTのものと同じスプール機能を行なう．またMFTと同様に，複数のリーダとライタがMVTに存在しうる．2つのシステムにおける唯一の違いは次の点である．すなわち，MFTではライタは(そして時々リーダも)システム・パーティションの中にあり，MVTではリーダとライタはオペレータがこれらのシステム・タスクを起動したときに動的に割当てられるリージョンの中にある点である．

MVTのイニシエイタ・タスクはMFLで述べたのと同じジョブ・クラス別方式を用いてユーザ・ジョブを主記憶装置中にスケジュールする．とりうる15のジョブ・クラスのうちの1つがユーザ（プログラマ）によって各ジョブに割付けられる．ジョブがシステムに入ったとき，それはジョブ・クラスに応じた待ち行列の中に置かれる．

ジョブ・クラスの*A*から*O*の文字自身は固有の順番の意味を持っていない．イニシエイタはMFTの場合と同様にジョブをスケジュールする．それは待ち合せ中のジョブを持っている最高の優先度のジョブ・クラス待ち行列から“最良”の候補を選択する．“最良”の候補はJCL EXECあるいはJOBカードにパンチされたPRTYパラメータでユーザの指定したジョブ優先度(およびFCFSのタイブレーキング方法)を用いて決められる．MFTと同様，ジョブのスケジューリング優先度は0から13が割付けられ，13がジョブ・クラスの中で最高のスケジューリング優先度である．PRTYは各EXECカードに別々に指定できるので，同じジョブの異なったジョブ・ステップは異なっ

た優先度が指定されることができる．MFT と MVT のジョブ選択の過程は同じであるが，似ているのはここまでである．MVT イニシエイタはリージョン内にある．それがあるジョブ・ステップをスケジュールするとき，イニシエイタはそのジョブ・ステップのためにリージョンを割当てる．主記憶装置は動的にジョブ・ステップに(そしてジョブ全体にである必要はない)動的リージョン領域から割当てられる．この違いを明らかにするため，われわれはまず例として1つの動作中のイニシエイタを持つ MVT 構成を用いた主記憶装置（リージョン)割当て過程をたどり，次に複数のイニシエイタの場合を見てみよう．

図 8-3.1 はオペレータがリーダ・タスク，ライタ・タスク，イニシエイタを起動した後の主記憶装置を示す．リーダとライタは図のように動的リージョン領域の中のリージョンを割当てられている．イニシエイタはオペレータのコマンド START INIT によって起動され，ジョブ・クラス *C* と *D*(この順番で)と 52k バイトのリージョンの大きさが指定された．クラス *C* と *D* のジョブをスケジュールすることのできるイニシエイタが図のように 52k バイトのリージョンの中にロードされる．それは同じクラスのものを分けるためにジョブ優先度を用いて，クラス *C* の待ち行列の中からジョブをスケジュールし始める．クラス *C* のすべてのジョブが終了してから，イニシエイタはクラス *D* のジョブのスケジュールを始める．最初に選択されたジョブは JOB2 と名付けられているとしよう．それに割当てられる最初のリソースはそのデータ・セットであり，次にジョブの指定した REGION パラメータ(図 8-3.2)と等しいかそれより大きな大きさのリージョンが割当てられる．この例では JOB2 はそのすべてのジョブ・ステップを処理するのに 90k のリージョンを要求している．(ジョブ・ステップごとのリージョンは EXEC JCL 文で個々に指定されていないと仮定する.)

そのリージョンを割当てるために，イニシエイタはまず自分自身のリージョンを返却する．リンク・パック領域にあるイニシエイタ・ルーチンは核から

```
//JOB2 JOB 'PL/I COMPILE', CLASS=C, PRTY=5, REGION=90K
```

図 8-3.2　MVT ジョブ・ステートメントにおけるリージョン・パラメータ

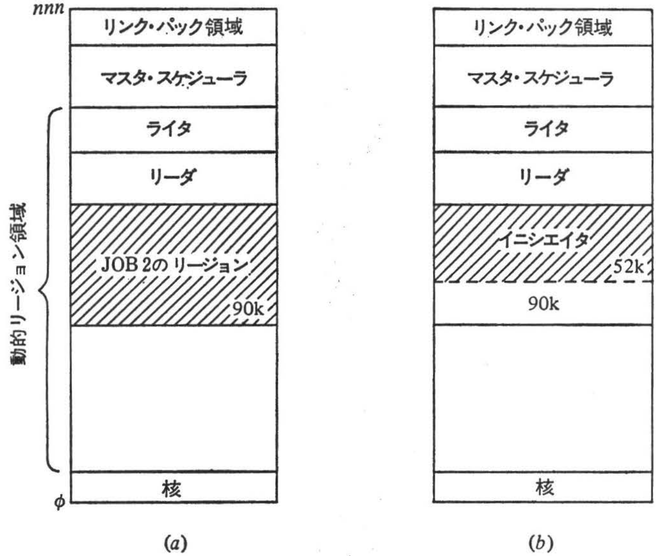


図 8-3.3 MVT リージョン割付けの

90k のリージョンを要求する. 核は主記憶装置の割当てを制御し, 動的リージョン領域の主記憶装置割付けのその時点の記録を持っている. 使用可能な連続した主記憶装置の最初の 90k がわれわれの例では JOB2 のリージョンとして割付けられる. これは図 8-3.3 の(*a*)に示されている. もし 90k の連続した主記憶装置が使用可能でなければ, イニシエイタは待ち状態に入る. ジョブの開始は 90k の連続した主記憶装置が使用可能となるまで引き延ばされる.

JOB2 にリージョンが割当てられた後で, 図 8-3.3 の (*b*) に示されるようにイニシエイタがその中にロードされる. イニシエイタは次に最初のジョブ・ステップに I/O 装置を割当てる. すべての装置が割当てられた時, LPA の中のイニシエイタ・ルーチンが JOB2 の最初のステップをユーザ・タスクとして, 割当てられたリージョンに結びつける. このユーザ・タスクによって要求されるプログラムがロードされ, ユーザ・タスクが実行に入る. 最初のジョブ・ステップが終了したとき, その 90k のリージョンは解放され, 新しい(52k の)リージョンがイニシエイタのために割当てられ, イニシエイタがロードされる. それから最初のジョブ・ステップのターミネイション処理が始まる. 最初のステップのすべての I/O 装置が返却され, イニシエイタは JOB2 の次のジョブ・ステップをスケジュールする. まず新たな 90k のリージョンを割当て,

次に I/O 装置を割当て，などと進められる.

動作しているイニシエイタがただ1つしかない例を見ると，リージョン割当てに不必要な動作が含まれているように思えるかもしれない．複数の動作中のイニシエイタを持つ例で，MVT のリージョン割当て手法がすぐに説明されるであろう．しかしまず MVT の主記憶装置管理法の付加的な性質を考えよう.

前の例で，ユーザ・プログラマは JCL ジョブ文でリージョンの大きさを指定した．あるジョブがおのおの 60k，90k，70k を要求する3つのジョブ・ステップを持つ場合，唯一つのリージョンの大きさでは少なくとも 90k 持たなければならない．この結果としてステップ1とステップ3の実行の間，30k と 20k のかけら(すなわち，リージョンの中の使用されない主記憶装置)がある．この記憶装置の無駄を避けるために，MVT はユーザが**各ジョブ・ステップ**のリージョンの大きさを JCL EXEC 文で指定することを許している．リージョンの中のかけらはユーザが彼のジョブ・ステップの大きさの要求量を知っている範囲で避けられる.

MVT での最小のリージョンの大きさはイニシエイタの大きさで決められる．前の例で見られた様に 52k が標準である．それゆえ，ジョブが(たとえば) 40k のリージョンを要求したら，核はジョブ・スケジューリングのためのイニシエイタをロードするための 52k リージョンを割当て，その結果，ジョブ・ステップ実行中 12k の無駄となる．イニシエイタ・ルーチンの多くは，特にターミネイション・ルーチンは**リエントラント**である．それゆえ，これらのルーチンはリンク・パック領域に置かれていると，複数のイニシエイタによって共有されることができる．もしこの手法が用いられると，最小のリージョンの大きさは 12k にまで減らすことができる．52k より小さなジョブを多く持つような装置では，LPA にリエントライトなイニシエイタ・ルーチンを置くことによって主記憶装置の使用率がよくなる.

あるイニシエイタはそれが割付けられたクラスのジョブをスケジュールするだけである．前の例では，イニシエイタはクラス C と D を割付けられていた．もしクラス C とクラス D の入力待ち行列の中にジョブがなければ，イニシエイタには仕事がなく，待ち状態に入る．しかし，それは主記憶装置を用いるリージョンの中に留まるのではない．イニシエイタのスケジュールするジョブがなく待ち状態に入ったとき，イニシエイタはそのリージョンを解放する．すべてのイニシエイタは LPA 中のあるリエントラントなルーチンにリンクされており，そのルーチンは新しいジョブがあるイニシエイタのクラスの待ち行列の1つに入ったとき，そのイニシエイタを再起動させる.

複数のイニシエイタが開始され，それぞれが割付けられたジョブ・クラスのジョブをスケジューリングするとき，MVT でのマルチプログラミングが行なわれる．MVT の動的な主記憶装置割当てを説明するため，簡略化した例を用いる．

1. 3つのイニシエイタが動作中である．
2. 各イニシエイタはおのおの違う1つのジョブをスケジュールしている．すなわち最初のイニシエイタはクラス A を，次はクラス D を，3番目のものはクラス G をスケジュールしている．
3. 各ジョブのリージョンの大きさは各ステップごとに指定されている．
4. 主記憶装置の動的リージョン領域はジョブ・スケジューリングのため 400k が使用可能である．
5. イニシエイタの大きさ，したがって最小のリージョンの大きさは 52k である．
6. システムでは図 8-3.4(*a*)に示すように現在3つのジョブ・ステップが実行中である．

図 8-3.4 の(*a*)で示されている3つのリージョンにはジョブ名，ステップ名，(括弧の中に)ジョブ・クラスが書かれている．斜線の領域は主記憶装置の使用されていない部分を表わす．(*b*), (*c*), (*d*) では三つのジョブのために新しいジョブ・ステップがスケジュールされたときに主記憶装置に起る3つの変化を示している．

(*b*)では JOB4 の STEP2 が終了し，120k を要求する JOB4 の STEP3 がクラス A のイニシエイタによってスケジュールされたと仮定する．STEP 2 が終了したとき，MVT はその 70k のリージョンを解放し，(動的リージョン領域の先頭の) 52k のリージョンをクラス A のイニシエイタのために割当てる．イニシエイタ・ターミネイタ・ルーチンは STEP2 で用いられた I/O 装置をすべて解放する．イニシテイタは次に JOB4 の次のジョブ・ステップ(STEP3)をスケジュールする．もし JOB4 がもうステップを持っていなければ，イニシエイタはクラス A の待ち行列から次の“最良”のジョブを選択する．STEP3 がスケジュールされた時，クラス A のイニシエイタはまずステップのリージョンの大きさ(REGION = *nnn*)を調べる．(*b*)の例では，STEP3 は 120k を必要とする．イニシエイタは自分のリージョンを返却し，(*b*)のように 120k のリージョンを割当てる．(MVT は上から下へ探すが，この場合は十分な大きさの連続した領域は動的リージョン領域の一番下にある．)

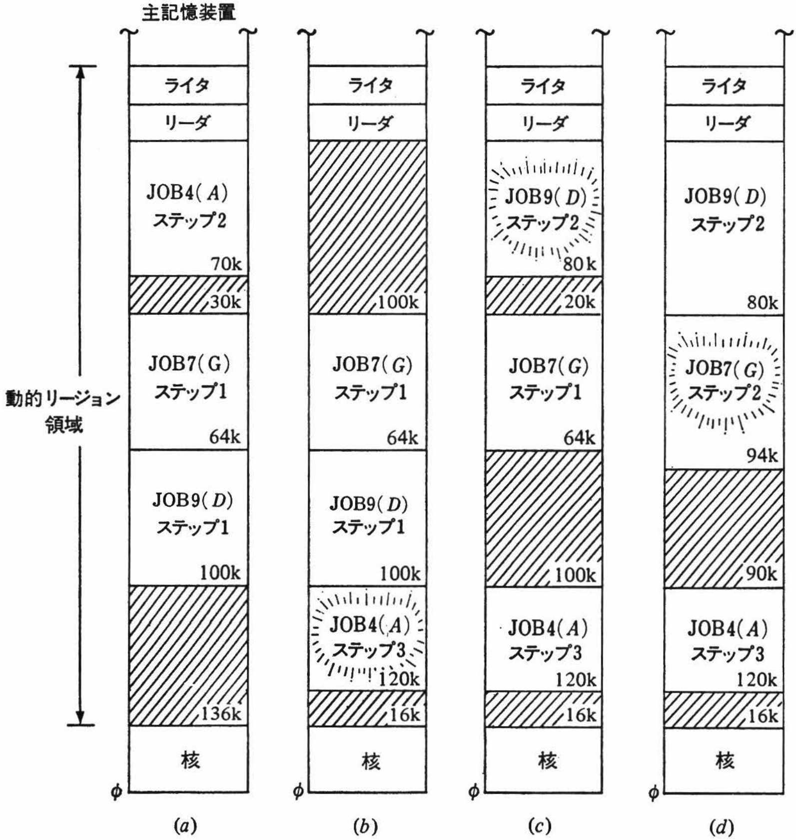


図 8-3.4　MVT のマルチプログラミング例

(c) では JOB9 の STEP1 が終了し，JOB9 の STEP2 がクラス D のイニシエイタによってスケジュールされたと仮定する．主記憶装置の使用されていない大きなかたまりがあるので，動的リージョン領域の上端にその 80k が割当てられた．(d) では JOB7 の新たなステップがスケジュールされたときのシステムでのもうひとつの推移が示されている．

これらのすべての例で，主記憶装置のかけらがあるが，そのかけらは MVT が新たなリージョンを割当てるごとに変化している．3 つのイニシエイタしか動作していないので 4 つ以上のリージョンが割当てられることはない．かけらの大きさから，おそらくもう 1 つのイニシエイタが起動されて，マルチプログラミングのレベルを増すことができることがわかる．15 までのイニシエイタが

MVT では起動でき，それぞれに異なった記憶装置の保護キーが割付けられる．

MVT はその主記憶装置管理手法をさらに動的にするオプションを持っている．そのオプションは**ロールアウト-ロールイン**(rollout-rollin)と呼ばれる．ロールアウト-ロールインを用いて，ジョブ・ステップ実行中にリージョンは動的に大きさを増すことができる．MVT 装置はどのジョブ・ステップが他のジョブ・ステップを追出すことができるか，そして，どのジョブが追出されてもよいかを指定しなければならない．“特権を与えられた”ジョブ・ステップは主記憶装置を動的リージョン領域からさらに要求することによって，実行中にそれ自身に“付け加え”てそのリージョンの大きさを増すことができる．もしジョブ・ステップの要求を満たし，そのジョブ・ステップのリージョンに連なった空いた空間が使用可能ならば，その空間はジョブ・ステップによって解放されるまでそのリージョンに結びつけられている．もしあるリージョンが“特権を与えられた”ジョブ・ステップに隣接していて，そしてそれが追出されてよいなら，そのリージョンのジョブ・ステップは割込まれ，直接アクセス記憶装置に格納され，そして“特権を与えられた”リージョンが大きさを拡げる．追出されたジョブ・ステップのリージョンが再び使用可能となったとき，ジョブ・ステップは引戻され実行を再開する．MVT は静的再配置システムであるから，そのジョブは主記憶装置の同じ領域に戻らなければならない．ロールアウト-ロールインはピーク負荷時の遠隔処理のジョブのように，一時的に増加した主記憶装置を必要とするジョブ・ステップにとって好ましい特徴である．

MVT のジョブ管理手法はどのジョブ・ステップ(ユーザ・タスク)が主記憶装置に駐在するかを決定する．タスク管理(ディスパッチング)は(主記憶装置に駐在するすべてのタスクの中の)どのシステム・タスク，あるいはユーザ・タスクが CPU の制御を得るかを決める．MVT における CPU ディスパッチングはシステムの核のタスク・ディスパッチャによって制御されている．MFTと同様，標準のディスパッチング規則は HSFS である．しかし，MFT のパーティションと異なり，MVT のリージョンは(システム設計によって)固定のディスパッチング優先度を持たない．MVT では，ユーザは(一般に装置からの助言によって)システムに入る各ジョブ・ステップのディスパッチング優先度を指定する．

ディスパッチング優先度は JCL パラメータ (PRTY =) で割付けられる．あるジョブのすべてのジョブ・ステップが同じディスパッチング優先度を持つ

とき，JOB 文の中の **PRTY パラメータ**が用いられる．しかし，EXEC カードで各ジョブ・ステップに別々に PRTY パラメータを指定することができる．PRTY 値の2つの使われ方は (1) ジョブ・スケジューリングで同じクラスのジョブを分けるため，と (2) CPU ディスパッチングの優先度として，である．

MFT の手法と比較されるとき，ユーザにジョブのディスパッチング優先度を指定するのを許すことは歴然とした利点がある．ユーザはすべてのシステム・リソースを高い率で使用するために，I/O-バウンド・ジョブには高いディスパッチング優先度を，C-バウンド・ジョブにはより低い優先度を陽に指定できる．もちろん，これにはユーザが自分のジョブの性質を知っているとしている．しかし，もし PRTY パラメータが JOB 文にのみ許されていたとすると，2つの問題が起こる．

1. PRTY パラメータはジョブ・スケジューリング優先度とディスパッチング優先度の両方に用いられるので，装置が，高いジョブ・スケジューリング優先度と低いディスパッチング優先度の両方を要求するジョブを持つかも知れない．たとえば (スケジュールの) 高い優先度の C-バウンド・ジョブ．
2. 装置は一般に，あるジョブの一部のジョブ・ステップは高いディスパッチング優先度を要求し，また他のジョブ・ステップは低い優先度が割付けられるべきであるようなジョブを持っている．

これらの理由から，MVT は JCL EXEC 文の PRTY パラメータでジョブ・ステップごとのディスパッチング優先度を許している．複数のジョブ・ステップが同じディスパッチング優先度を持って同時に実行している時，FCFS 規則がそれを分けるために使われる．

MVT ユーザはジョブ(あるいは，ジョブ・ステップ)にディスパッチング優先度を割付けるので，そしてその MVT ユーザはパーティションの大きさに“合った”ジョブかどうか気にしないので，MVT は MFT より操作し易い．MVT の動的主記憶装置制御で，MVT オペレータが主記憶装置割当てを再定義する必要は MFT より少ない．MFT におけるような適当なディスパッチング優先度でジョブをパーティションに合わせるという問題は，MVT ユーザがディスパッチング優先度を割付けるので，MVT においては除かれている．

MVT の設計に組込まれた HSFS ディスパッチング規則に加えて，MVT 装置ではまた**タイム-スライス** (time-slice) オプションを使える．ある選ばれた

ディスパッチング優先度(それは導入設置時に指定される)を持ったすべての実行中のユーザ・タスクはラウンド・ロビン・スケジュール規則(第4章参照)にしたがって取扱われる．タイム・スライスされる優先度レベルより上や下のすべてのユーザ・タスクはHSFSを用いてスケジュールされる．CPU時間がタイム・スライスされたタスクに使用可能であるとき，おのおのの動作可能なタスクは，より高い優先度のタスクによって割込まれるか，あるいはすべてのタイム・スライスされたタスクが待ち状態に入るまで，ラウンド・ロビンでCPU時間の一定量が与えられる．後者の場合，ディスパッチャは(HSFS規則を用いて)より低い優先度のタスクを選択する．タイム-スライス・オプションはまたOS/MFTでも使用可能である．第4章で論じられたように，タイム・スライシングを用いたラウンド・ロビン・スケジューリングは，短いC-バウンド・ジョブに対しては応答時間の際だった改善が可能である．

パーティション当り1つのユーザ・タスクしか許さないMFTの標準バージョンとは異なり，MVTのユーザ・タスクはそのリージョンにサブタスクを付け加えられる．この機能は“ATTACH マクロ”によってなされる．サブタスクが生成されたとき，ユーザ・タスクはその主記憶装置要求とディスパッチング優先度(これは親ユーザ・タスクより大きくあってはならない)を指定しなければならない．サブタスクのためにそのリージョンの中から主記憶装置が割当てられる．サブタスクのディスパッチングのためにタスク制御ブロック(TCB)が生成され，プログラムがサブタスクの主記憶装置の場所にロードされる．一旦起動されると，ユーザ・タスクとサブタスクの間でコミュニケーションが行なわれ，データが受け渡されることもあるが，サブタスクはリージョンの中で独立に働く．ユーザ・タスクがそのリージョンの中に各サブタスクに必要な主記憶装置を持っていれば，それは複数のサブタスクを付け加えてもよい．それでリージョンの中に，ジョブ・ステップは数個の並行機能を生成できる．このことは回線制御とメッセージ処理の間で並行性が得られている遠隔プロセシング制御プログラムで特に有効である．プログラマがアセンブリ言語，またはPL/1を用いる場合にだけ，マルチタスキングがジョブ・ステップの中で行なわれる．

第7章の導入部で，今日のオペレーティング・システムは複雑で巧緻であるように思われると述べたが，それらの機能の一部はまだかなり初歩的である．MFTとMVTは共にジョブ・スケジューリングとCPUディスパッチング機能を持っているが，実際上ユーザは自分自身のジョブの性質以外のジョブの性質のかなりの理解を必要とするスケジューリング上の多くの判断をさせられ

タイプ	目的	優先度規則	
ジョブ・スケジューリング	ジョブ待ち行列からジョブを選択し主記憶に転送する	*PCP*:	ジョブ・クラス，優先度はない．ジョブは入った順に処理される
		MFT:	1. ユーザは(ジョブ・カードで)各ジョブに CLASS = {*A* から *O*}　イニシエイタ起動時オペレータによってタイプ入力される順番により文字につけられた優先度の意味が決められる． PRTY = {0から13} 13 = 最高優先度 0 = 最低優先度を割当てる 2. 主記憶の各パーティションには3つまでのクラスが SYSGEN 時或いはオペレータによって割当てられる 3. あるパーティションのジジョブ・イニシエイタは最高の CLASS ジョブを選択する．同クラスのタイブレーキングは PRTY によってなされる．同 PRTY のタイブレーキングは FCFS による．
		MVT:	1. ジョブ CLASS, PRTY は MFT と同様にユーザによって割当てられる 2. (オペレータによって起動される)ジョブ・イニシエイタはそれがサービスできるクラスを指定する 3. イニシエイタが同クラスのものを PRTY によりタイブレーキングする．同 PRTY のタイブレーキングは FCFS による．
ディスパグッキン	主記憶中にあるジョブの中から CPU 実行のためのジョブを選択する	*PCP*:	ディスパッチングなし(主記憶中には高々1つのジョブしかない)
		MFT:	パーティション番号 *P*0 が最高優先度で，*P*1 が次，…等
		MVT:	リージョンの間でのディスパッチングには PRTY だけが用いられる．

注：1．ジョブ・スケジューリングは横取り方式で**ない**(未終了のジョブに新しいジョブがとってかわることはない)．
　2．ディスパッチンはグタイム・スライス・オプションを除いては事象駆動型(割込みが生じたときにイニシエイトされる)である．ディスパッチング優先度は各ジョブ・ステップごとに EXEC カードの DPRTY を用いて指定してもよい．

図 8-3.5　OS/360 ジョブ・スケジューリングとディスパッチングの要約

る．これは図 8-3.5 にまとめられている．これらのシステムはジョブ・ステップの性質が変るにつれてジョブ・ステップ実行中に動的に優先度を変えることはできない．たとえば，処理の前の部分のジョブ・ステップは I/O-バウンドであり，後では C-バウンドである場合がある．このような要求の結果，OS 強化のために（OS システム開発というより）“ユーザ開発”の機能がもたらされた．そのような機能の１つの例は**ヒューストン自動スプーリング・システム**（Houston Automatic SPOOL ing System 略して **HASP**）である．

8-4 HASP 概略［RY 参照］

HASP は IBM で開発されたが，一般のシステム開発の過程でなく，ヒューストンの“ユーザ・グループ”によって開発された．HASP は次の２つの主な機能で MFT と MVT を強化している．

1. 標準 OS/360 よりシステム入力，システム出力をうまくスプールする．
2. I/O と CPU の動作をモニタし，その時したがっているディスパッチング優先度を動的に調節するディスパッチング・オプション（これは OS/VS システムでも使用可能である）を持ち，I/O-バウンド・ジョブ・ステップには，その時の C-バウンド・ジョブより高い優先度が与えられる．

MFT あるいは MVT と共に用いられるとき，HASP は最高の優先度のシステム・タスクとして実行される．HASP スプーリング機能は２つの基本的な理由から OS リーダやライタが効率よく働く．(1) HASP リーダはシステム入力を読む時に JCL を解釈しない．(2) HASP はその I/O を行ない，入出力のデータ・セットのためのバッファ空間を標準の OS 手法より効率的に管理する．

HASP はすべてのシステム入力と出力を処理し，OS のインタフェースとして動作する．HASP は一時に一つのジョブを OS ジョブ待ち行列からスケジュールし，そこに返却する．その時，ジョブの JCL が解釈され，実行のためスケジュールされる．JCL の解釈を後回しにすることは HASP のシステム入力速度を向上させる．その欠点はユーザに JCL エラーを知らせるのが遅れることである．

HASP は I/O による制限のあるタスクに高い優先度を与えることによりハードウェア・リソースの使用率の向上を図った，**発見的ディスパッチング**と呼ばれるディスパッチング・オプションを持つ．使用に際して HASP 装置は，どのジョブ・ステップが発見的ディスパッチャによって制御されるかを指定し

なければならない．MVT ではディスパッチング優先度割付け，MFT ではパーティション割付けで行なわれる．実行中，これらのジョブ・ステップは**発見的ディスパッチング・グループ**の中に置かれる．OS タスク・ディスパッチャがこのグループに制御を渡したときに，HASP はどのジョブがディスパッチされるかを制御する．

発見的ディスパッチングは，オペレーティング・システムがジョブ・ステップのある性質をその実行中にモニタできるはずであるとの仮定に基づいている．特に，発見的ディスパッチャは，あるジョブが現在 I/O–バウンドされているのか C–バウンドされているのかを検知する．ディスパッチング・グループの実行可能なタスクがディスパッチされたとき，そのタスクは CPU 時間のある単位量が与えられる．もしそのタスクがこのタイム・スライスをすべて用いたら，そのタスクは C–バウンドであると見なされ，CPU サブグループに入れられる．しかし，もしタスクがタイム・スライスが終る前に I/O 動作のため待ち状態に入ったら，それは I/O サブグループに置かれる．発見的ディスパッチャはまず I/O サブグループをスケジュールし，そして，もしすべての I/O タスクが待ち状態に入ったら CPU サブグループをスケジュールする．I/O サブグループの任意のタスクは CPU タスクを横取りできる．その結果としてできるだけ多くの I/O を動かし，残りの CPU 時間をラウンド・ロビン方式で C–バウンド・タスクにディスパッチするアルゴリズムとなる．タスクの性質が C–バウンドから I/O–バウンドへ，またその逆に変るにつれて，HASP はそのサブグループを変更する．たとえば，もし I/O サブグループのタスクがそのタイム・スライスを全部使ったら，そのタスクは CPU サブグループの先頭に置かれる．これらの変化はジョブ・ステップの実行中動的に起こる．

実行しているタスクのモニタリングに加えて，発見的ディスパッチャは使われているタイム・スライスの効率もモニタする．導入設置時に決められた時間間隔（タイム・スライス自身よりずっと長い）ごとにディスパッチャは I/O タスク（と制御されているすべてのタスク）の比を，やはり導入設置時に指定されている I/O タスクと CPU タスクの望まれる混合比と比較する．もし充分な I/O タスクが見つけられなければ，タイム・スライスは大きくされる．見つけられる CPU タスクの数を増加するためには，タイム・スライスが小さくされる．タイム・スライス調整は上限と下限以内で行なわれる．HASP の発見的ディスパッチャは実際の動作負荷が I/O 型と CPU 型のジョブの混在したものであるような環境で最も効果的である．発見的ディスパッチング機能の基本的な目的はシステム・スループットの改善である．

HASPはユーザの要求からでてきたオペレーティング・システム高度化の1つの例である．数多くのオペレーティング・システムに対してこのようなことが多く行なわれてきた．それはオペレーティング・システムの詳細なスケジューリング，リソースの制御，動作負荷要求に対するリソース利用率の均合などをより自動的に行う方向に向けられている．これによりユーザはシステムの目標，方針の設定，システム性能のモニタリングなどの(詳細に述べないが)重要な仕事に注意を向けるようになる．

8-5 ソフトウェア・モニタリング [CA 1, IB 7, IB 8, IB 9 参照]

OS/360は動作分析の2つの基本的なレベルのソフトウェア・モニタリング(第7-6節参照)を供給する．それらは2つのモニタリング・プログラムに対応する．すなわち，(1) **システム管理機構**(SMF：system management facility), (2) **一般化トレース機構**(GTF: generalized trace facility)である．両方の手段はオプションである．それらが使用されるとき，いくらかの主記憶と直接アクセス記憶装置を要求する．SMFはジョブとジョブ・ステップの段階でデータを集める．そしてCPU時間，経過時間，I/O操作の回数であらわしたI/O動作の総量などのデータが記録される．SMFデータはジョブ会計に用いられ，統計的整理をして動作負荷解析に用いられる．そのデータはまたいろいろなコンパイラ使用のような動作負荷を，内容，機能，ジョブ・ステップの頻度などについて調べるためにも用いられる．SMFプログラムがOS/360で用いられる時，システム生成時にシステム核に組み込まれ，核の大きさが増す．データの収集に2つの直接アクセス・データ・セットが要求される．データ収集とは別に，SMFはまた装置がそれ自身のルーチンをOS制御プロセスの指定の場所に挿入することを許している．

GTFはシステム事象を動作分析のより詳細なレベルで——システム/360/370では割込みレベルで記録する．装置は1つまたはそれ以上の割込みタイプを，あるいはトレース期間のすべての割込み関係の事象を，選択的に記録でき，トレースの記録を内部テーブル(主記憶装置)に，あるいはテープまたはディスクの二次記憶装置に蓄えることができる．GTFは (1) プログラムまたはシステム・エラーのデバッギングと，(2) プログラムまたはシステムの性能の細かな調整のためにこのデータを操作するようにし向けられる．システムまたはプログラム・エラーに先立つ事象のトレースは，エラーの原因の検知に非常な手助けとなる．**内部トレース・モード**では，GTFは最新の割込み関係の事象をすべて主記憶装置中のテーブルに記録する．テーブルの大きさは記録される事

象の数を決定し，最新の情報を保持するために**ラップ-アラウンド**(wrap-around)手法が用いられる．もしシステムの誤り(またはユーザ・プログラムの誤り)が起きたら，テーブルが調べられ，その前のシステムの動作が再構成される．

GTF **外部トレース・モード**のでの動作が性能測定の研究のためにはもっと興味深い．トレースの記録はI/Oバッファに収められ，バッファが一杯になった時，テープまたは直接アクセス記憶装置に転送される．前に述べたように，バッファの大きさ(と使われるバッファの数)はトレース情報が失なわれないように充分大きくなければならない．注意すればバッファ空間と失われる情報との思慮深いバランスをとれるだろう．I/O 割込みに関するすべての事象に対して選択的トレースを行うことができる．そして，たとえば，特定の SVC に関する事象のみをトレースし，他のすべての SVC を無視して，ある割込みのクラスの選択がなされる．それで以下のような選択的な研究がなされる．

1. どの SVC を主記憶常駐とするかを決めるために，最も頻繁に参照される SVC を見つける．
2. データ・セットや装置の利用率を測る．
3. I/O 動作のチャネルでの分布を測る．

もしすべての型の割込み事象が GTF トレースで集められたら，そのデータは OS シュミレータを駆動する入力として用いられるであろう．GTF のような道具は動作中の OS システムの多くの側面を研究する機会を与える．使用時には，GTF はそれ自身のパーティション，あるいはリージョンと少しばかりの CPU 時間を必要とする．GTF によって得られるデータは，要求される型の解析をする適当なプログラムを書くことによって解析される．

8-6 OS/360 におけるデッドロック回避 [HAV, IB3 参照]

この節では7-6節で述べた一般的な原理のいくつかについて IBM OS/360 MVT オペレーティング・システムの場合で説明する．

たとえば終りのないループのプログラムによる CPU の閉塞を防ぐため，OS/360 には

TIME = (5, 20)

のようなオプションのキー・ワード・パラメータがあり，各ジョブ・ステップごとに，すなわち，EXEC ジョブ制御文(ジョブ全体のときは JOB 文)で指

定できる．上の指定の例はもし現在のジョブ・ステップ(またはジョブ)が **5** 分以上，CPU 時間で 20 秒以上使ったら，そのジョブ・ステップが一部となっている全体のジョブが終了されることを示している．この方法で指定できる最大時間は 1440 分(24 時間)以下である．CPU 時間制限に加えて，OS は自動的にジョブが WAIT 状態に連続して 30 分以上も入っていないかをモニタしており，その場合システムは自動的にそのジョブを無効とする．非常に長いジョブが CPU 時間，あるいは WAIT 時間の制限を正当に超えることが(非常に稀ではあるが)考えられるので，ユーザは TIME = 1440 と指定すれば CPU 時間と WAIT 時間の両方の制限をすべて無視することになる．最後にもしユーザが TIME パラメータを全く省略したら，(システム・プログラマによって設置時に指定された)入力リーダの中のパラメータ・リストによって指定のない場合の制限値が自動的に与えられる．

もし CPU 時間が各ジョブ・ステップに対して指定されていて，1 つでもその制限を超えたなら，ジョブ両体が無効とされることに注意してほしい．

IBM OS システムでのディドロックを取扱うための主な対策は J. W. Havender によって書かれている[文献 HAV]．彼はまた設計の決定の陰に隠れている多くの原因を解き明かした．以下の議論は Havender の扱いにぴったりと沿っている．

OS/360/370 での用語では**ジョブ・イニシエイタ**(しばしば単に**イニシエイタ**)と呼ばれるプログラムが各ジョブ・ステップを開始させる．一旦開始されたら，そのステップは**タスク**と呼ばれる．イニシエイタはステップの開始前にステップのためのあるリソースを獲得する．後で述べるように，あるリソースは全体のジョブの開始時に獲得され，またあるものはジョブ・ステップの途中で獲得される．ここしばらくの間はジョブ-イニシエイタの機能だけを考えよう．イニシエイタの数個のコピーが同時に動作できるので，すなわち，開始され終了していないので，OS はマルチタスキング・システムである．ジョブとジョブ・ステップによって要求されるリソースは次のように分類される．

1. 装置：テープ，カード読取機，プリンタなど．
2. データ・セットジョブ：制御言語(JCL)の名前付けの部分で名付けられた直接アクセス記憶装置のリージョン．
3. 主記憶空間

順次**再利用可能な**リソースは，あるタスクがそのリソースを返却するまで高々 1 つのタスクによって使われる．これらのリソースは OS のマクロ命令 ENQ

と DEQ によって要求され返却され，これにより“排他使用”が励行され，リソースの待ち行列が維持される．

順次再利用可能なリソースの物理的性質から，装置は横取りが不可能であると考えられる．ジョブ・ステップが動的に装置を割当てることを許すと，その結果デッドロックがありうる．そこで，OS は各ジョブ・ステップが必要なすべての装置を宣言することを要求し，そのステップはその装置がすべて得られるまで開始されない．装置のこの扱いはデッドロック回避策にしたがっていると考えられる．イニシエイタが実際に装置を割当てている間，装置が排他利用リソースであるから，オペレーティング・システムは他のイニシエイタが同様のことを行なうのを締め出している．各ジョブ・ステップの終りには，それに割当てられたすべての装置が解放される．

データ・セット・リソースの獲得は装置とは少し違った問題である．データ・セットはその物理的性質から数個のジョブ・ステップにより同時にアクセスされうる．（この可能性は DD JCL 文の SHR オプションでユーザにより宣言されるであろう．）しかし，ほとんどのデータ・セットはそのようには使われない．データ・セットが1つのジョブ・ステップの間でグローバルであることを許し，しかもこの時他のジョブによるジョブ・ステップからのデータ・セットへのアクセスを防ぐことが最も望ましい．もしデータ・セットが排他利用の強制と組合せてジョブ・ステップ単位に割当てられたら，デッドロックが起こりうる．たとえば，ジョブ1の最初のステップがデータ・セット A を割当て，次のステップが B を割当てるとしよう．またジョブ2が最初のステップで B を，2番目のステップで A を割当てるとする．2つのジョブはそれらのデータ・セットを逆の順に要求しデッドロックを可能とするように見える．ジョブがそのすべてのジョブ・ステップにわたってデータの排他利用を続けることを許し，しかもデッドロックを避けるために，OS はジョブの開始前にそのジョブに必要なデータ・セットをすべて獲得する(各ジョブ・ステップで獲得され，解放される装置の場合とは異なる)．これはジョブのジョブ制御文をすべて調べ，(DD カードの情報から) そのジョブのために宣言されたすべての SHR でないデータ・セットのリストをコンパイルすることでなされる．そのジョブはすべてのデータ・セットが獲得されてから開始される．この方策はデータ・セットのデッドロックを避けるように思われる．不幸なことに，OS はジョブ・イニシエイタによってだけでなく，またデータ・セットが動的に獲得されることを許しているので，デッドロック回避は水も漏らさぬものではない．この結果デッドロックとなりうる．

リソースの第3のクラスはREGIONと呼ばれるジョブ・ステップのための主記憶空間である．前にも述べたように，リージョンの大きさは各ジョブ・ステップごとに別々に(JCL の EXEC 文で)宣言されるか， さもなければ JOB 文で指定されたリージョンの大きさにジョブのすべてのリージョンの大きさが決まる．リージョンがジョブ全体でなく各ジョブ・ステップごとに割付けられるということは，あるステップが他のものより大きな空間を要求するかも知れないので，高価な主記憶リソースの割付けの柔軟性を著しく増す．

次に，どのようにジョブ・ステップが開始されるか，少し詳しく考えよう．まずいろいろな事が起こる**場所**を考えることによって足場を築き，それから時間順に事象の考察を進めよう．イニシエイタはジョブ・ステップを開始するのに必要なリソースの割付けと解放を行なう．イニシエイタはそれ自身プログラムであるから，空間を要求する．オーバレイ手法を用い，イニシエイタのほとんどをリージョンと同じ空間で実行することにより， OS はこの機能による空間のオーバヘッドを避けている．しかし，小さいが重要なイニシエイタの“種”となるモジュールはいつも主記憶装置のスーパバイザのリンクパック領域に常駐している．このように足場を決め，さてジョブが終ったとき何が起こるかを考えてみよう． まず， 今終了したタスクをオーバレイする(イニシエイタの一部である) ターミネイタ・ルーチンを呼びだす．ターミネイタは装置を解放し，そして次のジョブ・ステップを開始するためにイニシエイタを起動する．これには新たなジョブ・ステップに装置とリージョンの両方を割当てなければならない．どの順番でこれがなされるかは重要であろうか．もしある装置が使用できなければ，新しい未割当てのリージョン(すなわち主記憶装置)が拘束されないので，まず装置が割当てられるべきであると思われるかも知れない．これは割当て順序

装置 ⟶ リージョン

を示唆する．これがジョブタスク1になされたと仮定しよう．タスク1が終了したとき，ターミネイタは同じリージョンで実行され，次のジョブ・ステップは(われわれの仮定した順序で)まず装置割当てから始めなければならない．装置の割当てはかなりの主記憶空間を要求するので，空間のオーバヘッドを避けるために， OS はジョブ1のタスク2のための装置割当てに同じリージョンを使用するであろう．もし割当てが成功するとイニシエイタはリージョンを解放し，そのリンク・パック領域に退き，それからタスク2のためにリージョンを割当てるであろう．しかし装置割当ての間にタスク2がすでに割当てられてい

るとわかっている(たとえば)テープ装置を割当てようとしたと仮定しよう．さらにこのテープ装置を保持しているジョブがちょうど装置割当てを終えたところで，リージョン割当てをしようとしており，これが満たされるためにはジョブ・タスク2のリージョンの空間を使わなければならないとする．その空間は解放される予定であるが，テープの割当て待ちにひっかかっている．明らかにこれはデッドロックであり，その原因は実質的に逆の順に割当てていることにある．

ジョブ1．タスク2: リージョン ⟶ 装置
ジョブ2．タスク1: 装置 ⟶ リージョン

タスク2が仮定された順序を守っていないように見えるかも知れないが，上でわかったように装置を割当てるためにリージョンが必要とされる．これらの理由から，OS は実際には装置とリージョンの割当てを

リージョン ⟶ 装置

の順で行なっている．前にも述べたように，データ・セットは全体のジョブの開始時に最も早く割当てられ，結果としての順序は

データ・セット ⟶ リージョン ⟶ 装置

となる．これによりデッドロックは回避されたが，あるジョブ・ステップで装置より先にリージョンを獲得すると，装置が使用可能となるのをそのステップが待つので，かなり主記憶装置を遊ばせることになる．

演 習 問 題

1. (a) OS JCL の3つの主要なステートメントは何か．そしてそれらの主要な機能は何か．
   (b) 記憶装置-時間の割当てに影響を及ぼすような JCL パラメータを少なくとも4つあげ，簡単に説明せよ．
   (c) 次の問を考察し議論せよ．OS/360 JCL はどのような点で他のプログラミング言語と類似しており，どのような点で異なっているか？ これを JCL 構造の改善について論ずるために用いよ．
2. (a) カタログ化手続(CP)とは何か？
   (b) ジョブ処理においてシステムはどのように CP の参照を行なって

いるか簡単に述べよ.

3. ファイル名とデータ・セット名の違いを明らかにし，それらがどのように，また，いつ関係づけられるかを示せ.

4. OS/360 の機能の主要なクラスを識別し，簡単に記せ.

5. 次のものはどのような機能を持つか.
   (a) システム常駐部
   (b) 過渡的領域(SVC のための)
   (c) 過渡的領域の使用に代るものは何か.
   (d) それを用いることで性能にどのような影響がある.

6. OS/MFT と OS/MVT を次の点から比較せよ.
   (a) 主記憶装置割付け: 問題プログラムとシステム・プログラムの両者を考えよ.
   (b) ジョブ・スケジューリング
   (c) ディスパッチング(CPU の)

7. OS/360 で可能なマルチプログラミングの程度は一般に何によって決まるか.

8. OS/360 でプログラムが最終的にメモリに割付けられるのはいつか. このシステムでより後に割付けられるということは考えられるか. それはなぜか？ これが性能に及ぼす影響はどうであるか.

9. OS/MVT のマルチタスキングとは何か. なぜ，そしてどのような時にそれが望ましいか.

10. OS/MVT のリンク・パック領域での機能を記せ.

11. (a) MVT のロールアウト-ロールイン・オプションについて簡単に述べよ.
   (b) なぜそれが用いられるのか.
   (c) その限界，欠点は何か.

12. MVT タイム・スライス・オプションはどのような基本的なスケジュール・アルゴリズムを用いるのか. これはマルチプログラミング環境下で有効か.

13. HASP スプーリングは OS リーダとライタを持つスプーリングとどう

違うのか.

14. (a)　HASP の発見的ディスパッチング・オプションについて記せ
(b)　本文の前の方(4章)で書かれた基本的なスケジューリング・アルゴリズムとこれを対比せよ.
(c)　一般的な OS/MVT ディスパッチングとこれを対比せよ.

15. (a)　ハードウェア・モニタに比べてソフトウェア・モニタは何が主要な利点であり欠点であるか.
(b)　ソフトウェア・モニタの基本的動作を簡単に述べよ.
(c)　ソフトウェア・モニタの主要な使用法を少なくとも2つ掲げよ.
(d)　トレースの"レベル"とは何を意味するのか. OS/360 から例を示せ.

16. SMF と GTF でモニタされる事象のタイプを比較せよ.

17. 文献(1章, 7章の参照文献を手始めとすればよいだろう)を調べ, IBM のではない, 少なくとも1つのオペレーティング・システムのスケジューリング機能についてレポートを書け. それらを OS/MVT と比較せよ. 示唆 : CDC 6600, UNIVAC 1108, バローズ5500. (注 : 充分詳しいものは出版文献よりもマニュアルによって得られるであろう)

18. もしソフトウェア・モニタが利用できるのなら, その出力を以下のものに使え.
(a)　(例えば) 1カ月間のシステム活動状況の記述
(b)　**スループット**と平均経過時間の算出
(c)　チャネルと CPU の利用率の記述
(d)　各種コンパイラの利用率の記述
(e)　ジョブ・シミュレータを動作させるのに適当なジョブ記述子のストリームの生成

19. (a)　問18(d)のストリームで駆動されるシミュレータを書け.
(b)　その出力を問18(b)で得られたものと比較せよ.

参考文献

1. [BR] Brown, G.D.: "System/360 Job Control Language," John Wiley and Sons, New York, 1970.

2. [CA1] Campbell, D.J. and W.J. Heffner: Measurement and Analysis of Large Operating Systems During System Development, *AFIPS Conf. Proc.*, vol. 33, part 1, pp. 903-914, 1968 FJCC.
3. [CL] Clark, W.A.: The Functional Structure of OS/360 Part III Data Management, *IBM Systems Journal*, vol. 5, no. 1, pp. 30-51, 1965.
4. [FRE] Freeman, D.N.: A Storage Hierarchy System for Batch Processing, *AFIPS Proc.*, vol. 32, pp. 229-244, 1968.
5. [HAV] Havender, J.W.: Avoiding Deadlock in Multitasking Systems, *IBM Systems Journal*, vol. 7, no. 2, pp. 74-84, 1968.
6. [IB1] IBM Corp: "IBM System/360 Operating System Introduction," No. GC28-6534, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
7. [IB2] IBM Corp: "IBM System/360 Operating Systems Feature Guide," No. GC28-6716, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
8. [IB3] IBM Corp: "OS/360 Job Control Language Users Guide," No. GC28-6703, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
9. [IB4] IBM Corp: "IBM System/360 Operating System: MFT Guide," No. GC27-6939, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
10. [IB5] IBM Corp: "IBM System/360 Operating System MVT Guide," No. GC28-6720, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
11. [IB6] IBM Corp: "IBM System/360 Operating System Planning for Rollout/Rollin," No. GC27-6935, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
12. [IB7] IBM Corp: "IBM System/360 Operating System Management Facilities Reference Manual," No. GC28-6712, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
13. [IB8] IBM Corp: "OS/VS Service Aids Manual," No. GC28-0633, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
14. [IB9] IBM Corp: "AMAP (Advanced Multiprogramming Analysis Procedure Service Description Manual," No. GH20-0725, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
15. [IB10] IBM Corp.: "OS/360 Job Control Language Reference," No. GC28-6704, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
16. [IB11] IBM Corp: "IBM System/360 Operating System PL/I [F] Programmers Guide," No. GC28-6595, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
17. [ME] Mealy, G.H.: The Functional Structure of OS/360 Part I Introductory Survey, *IBM Systems Journal*, vol. 5, no. 1, pp. 3-11, 1965.
18. [RY] Ryder, K.D.: A Heuristic Approach to Task Dispatching, *IBM Systems Journal*, vol. 9, no. 3, pp. 189-198, 1970.
19. [WI] Witt, B.I.: The Functional Structure of OS/360 Part II Job and Task Management, *IBM Systems Journal*, vol. 5, no. 1, pp. 12-29, 1965.

第9章

タイムシェアリング・システム

タイムシェアリング型計算機システムは1960年代初期にいまの形で始めて出現した．それ以前の軍用“司令制御”システムや航空予約システムもある点ではそれと関連があるけれども，現在は通常**タイムシェアリング**とは次のような多数のユーザを有するシステムを指す．すなわちそのシステムでは各ユーザは自分自身私用の汎用計算機を持っていると感じていて，その上で自分自身のプログラムを定義でき，また走らせることができるようなシステムを指す．ただしシステムは各ユーザの要求のいくつかにはマンーマシン・インタラクションに対して快適な気持を感じさせるのに十分な早さで応答しなければならない．より一般的な“バッチ・システム”と比較すると，タイムシェアリングは**人間**の計算機に対するインタラクションの仕方に相違がある．それはずっとより親密な方法なので，問題解決，技術設計，およびあらゆる型のプログラムの作成およびデバッグのような活動にとって非常に便利である．創造的な環境におけるこのより向上した能力によって，人的資源のより生産的な利用ができるので経済性が上り，また人間の創造性を向上させたいという課題に対して今後も限りなく挑戦してゆくことができる．

タイムシェアリングの正当性について，以上はユーザ側から見たものであるが，それに加えてリソースの効率の良さ(すくなくともその可能性について)に期待をする向きもある．その1つは，バッチ作業負荷と共存して，そのシステムに結合された多くのユーザから高い率で要求が出ると，CPU 利用率が改善されるであろうという期待がある．というのは多くのバッチ・システムでは，典型的に CPU 利用率は30ないし40％という低い率であるからである．しかしながら，この可能性はスワッピングの“オーバヘッド”のためにすくなくともその一部は相殺されるかもしれない(下記参照)．

中央処理装置の“タイムシェアリング”——タイムシェアリング・システム

という一般的な名称はそれに由来しているのだが――は，ますます**共有**するようになってきたいくつかのものの1つに過ぎない．その他のものには，**プログラム**の共有，特にトランスレータ・プログラム(インタプリンタとコンパイラ)，サブルーチン・ライブラリ，およびユーザのプログラムのみならず，主記憶装置や他の**装置**の共有が含まれる．人間が反応するたびに利用できるというこの近づきやすさは，タイムシェアリング・システムが巨大なコミュニケーション・メディアとなることを意味しており，それによってわれわれは他人の経験をすばやく知ることができる．

さて，タイムシェアリングに対する基本的な動機について触れたので，そのようなシステムのインプリメンティションのいくつかの特色について述べよう．1つの論点は――それは驚くほど簡単に解決されてしまったが――現代の中型-大型計算機システムの装備は，多くのタイムシェアリング・システムを走らすために，新たに装置を付加したり新たな特徴をもたせる必要がほとんどないということである。3つの欠くことのできないCPUの特徴，すなわち，**割込機構，リアルタイム・クロック，および記憶保護機構**は，1960年代に出現したほとんどの計算機にとってもはや普通であった．というのは，それらが，最近のマルチプログラム方式のバッチ・オペレーティング・システム†を効率的に走らせるためにも，また必要であったからである．タイムシェアリング環境にとって欠くことができず，また独特なものともいえる装置の1つは**端末**である．それには，今日典型的なものとして，テレタイプやIBM“ゴルフボール”タイプライタのように，電話回線に接続するための電子回路を備えたテレタイプライタ式の装置がある．この種の端末は比較的安価で，ユーザはある与えられた型の端末ならどのタイムシェアリング・システムにも接続することができる(もちろん，前もって料金支払の取決めがなされているとして)．したがって，ある1つの端末を，計算機言語やその他の便宜・設備に対する種々の要求に応ずるために，通常の電話設備を通じて，いろいろな時に，異なる遠隔地の中央処理設備に(ダイヤル)接続することは全く可能である．それゆえ，公衆広域電話システムがタイムシェアリング・システムの一部であることが多い．計算機所在地では，おのおのの入回線上の信号はモデム(変調器-復変調器)によって電気的に変換された後，通常単一のレジスタに多重化して取込まれ，それから主記憶装置のバッファ領域に分配される．システムのプログラム

† プログラムの動的再配置を可能にする回路の設備は効率を改善することができるが，必須のものではない．IBM S/360計算機では，Mod 67のみがこの特徴を持っていた．ほとんどのCDC, Burroughs, GE, およびその他のシステムは，再配置ハードウェアを持っていた(10-11節参照)．

は，そこでこれらのメッセージを処理するのである．この多重化機能には，**伝送制御ユニット**と一般に呼ばれている装置を必要とする．

タイムシェアリング・システムはかなり多数のユーザに同時にサービスすることでそのコストのもとを取らねばならない．典型的な中型計算機はおよそ50のユーザにサービスする．多数ということは，経済的に成り立ちうるために欠かすことのできないものであり，そしてまたそれは技術的に可能である．というのは，各個々のユーザはシステムにとって軽い負荷と考えられるからである．このことは，たとえその端末で長時間のセッションが開かれていてもなおいえることである．大部分の時間が通常思考やキイ入力に費やされているからである．もちろん，タイムシェアリングの1つの主要な目的は，プログラムの編集のようなある手厳しい人間の要求に対する迅速な応答である．これらは，普通，かなり速い計算機からみれば比較的少量の処理しか要求していない(この重要な問題点についてはさらに後述する)．

接続された多くのユーザは(全体として)，目下のプログラムとデータを保持するのにかなりの記憶装置を要求する．この記憶装置に対する要求は，通常，経済的に引き合う高速主記憶装置のサイズをはるかに越えている．ある時点ではほんの2～3人の(時にはたった1人の)ユーザによって要求された記憶空間が主記憶装置中に保持されているだけで，残りのユーザのそれはドラムやディスクのようなより安価で，より遅く，より大容量の補助記憶上に保持しなければならない．システムがある特定のジョブを実行しようと決めたとき，もしそれが主記憶装置にないと(それはよくあることだが)，まず現在駐在しているあるジョブを主記憶装置から(たとえば)ドラムへ移動しなければならない(それによって，後からの処理の続行のために，その状態を保有しておく)．それから要求されたジョブがドラムから主記憶装置のちょうど今空いたばかりの部分に送られる．この処理は**スワッピング**と呼ばれ，これはたいていのタイムシェアード・システムの特性である．スワッピングはオペレーティング・システム・プログラムによって自動的に行なわれる．ジョブのサービスの順序の決定――それはどのジョブをいつスワップするかを決める――は，4章で議論された型のスケジューリング原理に従うこともできる．またその他の多くの規則が開発されておりこの章ではそれらのいくつかを調べる．そのようなすべての規則は1つの基本的な必要条件を満たさなければならない．その条件とはマン-マシン・インタラクションにとって本質的な，処理要求に対する速い応答である．

用語について一言述べておく．通信計算機システムの議論で，ユーザとシステムの間の1回のインタラクションに必要な仕事はトランザクションと呼ばれ

るであろう．1つのトランザクションはわれわれが前に**ジョブ**と呼んだものにいくらか似ているが，しかし個々のユーザにとっては，(あるプログラムを働かすというような) 1つの“ジョブ”には普通多くのインタラクションが必要であり，したがって多くのトランザクションが必要である．

いくつかのタイムシェアリング・システムはその端末ユーザに単一のプログラミング言語のみしか提供していないが，そのような場合そのシステムのことを**専用**システムと呼ぶこともある．専用システムの例には，IBM の QUIKTRAN や，いくつかの APL および BASIC システムがある．アナリストによっては，**専用**という言葉を，タイムシェアリング・ユーザには1つ以上の言語が与えられてもよいが，同時にはバックグラウンド(バッチ)作業負荷が走れない，という意味に用いることがある．バッチ・モードのプログラムは，しばしば，より多くのマシン・リソースを要求することができるので，1つの望ましい特徴として，端末で使用される言語およびコンパイラがバッチ・システムで使用されるそれらと互換性のあることが望ましい．互換性が利用できないと，これは仕事の挫折の大きな原因となる．特にあるジョブを小規模なアルゴリズム開発段階から，大量のデータを扱う段階に展開する場合に然りである．趨勢としては明らかに，バッチ互換言語を有する多種言語(すなわち非専用)タイムシェアリング・システムに向っている．

上述したいくつかの基本的な点を要約すると，タイムシェアリング・システムの主な目的は，効率的，経済的および快適なマン-マシン・インタラクションにある．ハードウェア，プログラムおよびデータを含んだ共通のリソースを人間の反応時間という時間尺度で共有するには，各ユーザの所在地に，あるかなり独特な設備(端末および通信装置)を必要とする．通信のためには現存の電話設備をうまく利用することができるが，一方では，システム内のリソースの内部的管理の大部にわたる新たな方向づけ，すなわち通常のバッチ型とは異なるオペレーティング・システムを必要とする．かなり多くのタイムシェアリング・システムが，**マルチプログラミング**や**仮想記憶装置**というような，バッチ・システムにおいて使用されている，あるいは有用とされている多くの特徴を含んでいるが，それらはこの本の他のところで考究されている．これらは本来タイムシェアリング・システムにとって独特なものでもなく必須なものでもないので，この章では議論されないであろう．

9-1　ユーザの観点といくつかの結論［BRY, GO, SA 1, SA 2, SC3, SM 参照］

序文において，タイムシェアリングによるユーザの利益の主ないくつかについて述べた．ここで，そのような主張を支持する少なくともある組織だった証拠を与える1つの研究に着手しよう．この型の証拠は，手に入れたり解析するのがかなり困難である．というのは，それには**人間**の作業習慣がからんでおり，それは一般的な結論を引き出すのが最も困難な問題であるからである．シミュレーション言語 DYNAMO によってインプリメントされた建築事業のシミュレーション・モデルを用いた1つの研究が MIT で管理学科の66人の学生を対象に行なわれた［SA 1, SA 2 参照］．各学生には事業と市場の条件についてのある情報が与えられ，その収益を最適にするために(計算機上の)そのモデルを使用することになっていた．そのクラスは2つのグループに分割された．両者とも同一のモデルを使用するが，一方のグループは計算機をタイムシェアード・システムとして利用し，他方のグループはバッチ・システムとして利用する．学生の出来ぐあいと態度についてのデータは，アンケートや教師による問題解答の分析およびリソースの使用による通常の計算機課金によって，得られた．この実験の主な結果は次のように要約される．

1. 両グループの全コスト(システムとユーザの作業時間の和)にはたいした差がなかった．しかしそれらコストは全く異った分布をしていた．タイムシェアード・グループでは，人的労力のコストはより低く，計算機のコストはより高かった．
2. タイムシェアリング・ユーザは，割当てられた問題に対して意義あるよりよい解答に達した．
3. バッチ処理グループでは，2倍以上の学生が割当てられた問題に対する有益な答を得なかった。
4. より多くの人がタイムシェアリング・システムを好んだ．

この実験は，測定容易な直接コストのみならず，よりよい問題の解答を得ることによる利益やユーザの態度を含めることの重要さを示している．他の類似の研究はもう1つ別の重要な結論を指摘している．すなわち，**ユーザの出来ぐあいのバラツキについては，グループ間でのそれよりもグループ内でのそれの方が大きかった．**これは，おそらく，人間の熟練の差が使用されるシステムの型の差よりもより大きい意義を持っていることを示唆している．

カテゴリ	典型的な料金	仮定の使用量	コスト/月
端末レンタル(IBM 2741)	$100/月	1	$100
電話データ・セット	28	1	28
電話賃借回線†	3/mile/月	50 mile	150
CPU 課金†	5/分	20	100
接続課金†	12/時	40	480
ファイル課金	1/7 kbyte/月	70 kbyte	10

† 接続課金は，典型的には
接続時間1時間につき CPU 時間1分
賃借電話回線の案分比例分担
を含む．

図 9-1.1 典型的な料金(1971)

個々のユーザの観点からみて見落すことのできない1つの重要な面に，ユーザが利用する設備とサービスに要するコストがある．**図 9-1.1** の一覧表は，コストの特定の値(これは疑いなく時とともに変る)というよりむしろどのようなカテゴリのコストがあるかに主として興味がある．端末レンタル，電話データ・セット，および電話回線コストについては，これ以上いう必要がない．CPU **料金**は，システムが個々のユーザのために作業した時間のみに適用される．接続料金はシステムに対して“使用開始の合図をした”ユーザに対して，システムを使用するしないにかかわらず，1時間ごと適用される．というのは，端末バッファ用の主記憶領域や通信回線制御用のサブチャネルのようなある種の中央計算機のリソースを拘束する必要があるからである．図 9-1.1 は種々の課金のカテゴリの率を与えている．コストはまた時間にも依存するであろう．典型的な時間では，コストの大部分は CPU 使用よりもシステムの通信部分のために費やされることになるかもしれない．このことは**電話回線集線装置**と呼ばれる装置を使うことを正当化する理由となっていることも多い．この装置は同一地方の多数の端末の信号を結合し，中央システムへの高価な賃借回線をより経済的に共有できるようにするものである．このことは，端末群と計算機との距離が長く(したがってコストが高く)なればなるほど重要となってくる．

さて，タイムシェアリング・システムにおける性能の論点を理解するために，ユーザのシステムに対する好み，すなわちユーザにとってシステムがどのようにみえたらよいかを考えよう．望ましい特徴の長々とした“洗濯物リスト”を作ることは比較的簡単であるから，その大部分はまた他のいかなる種類の汎用計算機システムにとっても共通であり，これはタイムシェアリング・ユ

ーザ特有の関心事をほとんど明らかにしないであろう．それよりこう質問をした方がよい．システムのいかなる性質がタイムシェアリング・システムのユーザにとってユニークなのか，あるいは少なくとも重要であるか？われわれは，手初めに，この章の序文で議論したいくつかの考慮を簡潔に再述することによって，それに答えよう．すなわち，**タイムシェアリング・システムの主要な目的は，迅速で，便利で，経済的なマン-マシン・インタラクションを提供することである．**この節の目的は，このことを出来るだけ明らかにし，いくつかの例によってそれを示すことにある．

最初に，ただ**一人の人**が端末によって，その人の“自家用”計算機とインタラクトしているような状況を調べよう．しばらくの間，多くのユーザがリソースを共有することに起因する性能上の問題には触れないでおこう（後節参照）．

たいていのタイムシェアリング・システムは会話型であるといわれていることが多い．というのは，ユーザ-マシン・インタラクションは“礼儀正しい”会話の形式をとり，それによって互いにメッセージを厳格に交互に送信すると考えられているからである．このことは，普通，伝送のためにユーザがタイプした行を受取った後は，システムが，その結果か，それ自身のメッセージか，時には単にキーボード解除信号かのいずれかで応答するまでは，キーボードを動かなくしてしまうようにすることによって，強制されている．

ユーザが必要とする主な型の機能は，**図9-1.2**にあげた．それらの機能は，ユーザが種々のサービス要求を行なうために出す**コマンド**を解釈するシステム側供給のプログラムによって，提供される．これらの機能についてのわれわれの理解をはっきりさせるために，それらとバッチ・システムのユーザに必要なものとを比較しよう．また，ユーザのプログラムを機械実行形式に変換する言語トランスレータは，バッチ・システムで普通見られるコンパイラ型のものと，最初は仮定しておこう．

図9-1.2の項目の大部分は，ユーザ・プログラムとデータはセッションとセッションの間ではシステム内に記憶されていなければならない，という必要性を暗示している．項目1と4は，ディレクトリの集合によってできている名付け機構で名付けられた対象に対して，普通適用される．異なるユーザによって同一の名前を付けられた対象の間での混乱を防ぐために，たいていのシステムではディレクトリの階層を保持している．事実上は，各ユーザの対象は，自動的にしかも暗黙のうちに，その名前の一部としてユーザの名前（あるいは識別子）を含んだものを有している．

プログラムとデータの編集はプログラムの形で普通供給されるが，これはユ

1. プログラム・データ対象
   (a) 現在名と記述のリスティング
   (b) 名前の割当と削除
   (c) 名前による対象の参照
2. 行番号と文脈によるプログラム編集(文とプログラム全体)
   (a) 表　示
   (b) 修　正
   (c) 削　除
   (d) 挿　入
   (注：修正は削除の後，挿入することによって行なわれる.)
3. データ編集
   (a) プログラムの入力データ
   (b) 計算結果のデータ
4. 合　成
   名付られた対象(プログラム，ファイル)を組み合わせて新たな名前をもつ対象を構成する.
5. 順序制御
   (a) 使用開始と使用終了(ログオンとオフ)
   (b) プログラムのスタート
   (c) プログラムの中断
   (d) 中断されたプログラムの再開
6. ユーザの誤りの検出
   (a) 入力中の各行毎
   (b) プログラム入力後
   (c) 実行中

図 9-1.2　望ましいユーザの機能のチェック・リスト

ーザにとっては項目2.に述べたオペレーションに対応する編集コマンドの集合としてみえる．ユーザ・プログラムの制御の下で入力しようとするソース・**データ**の編集は，プログラム編集用のものとほとんど同一の機能によって行なうことができる.

表にあげた要求を満たすにあたっての主な困難は項目5.――プログラムの中断可能性――に集中している．(ここでは，1人のユーザによって要求される中断可能性のみについて述べており，スワッピング機能のためシステムによって要求されるそれではないことを思い出そう.)これは本当に必須なものかど

うか質問することから始めよう．もしそうなら，基本的な必要性を満たす最小の機能は何であるか．プログラムを中断する能力は実際必須なのである．というのは，計算の理論によれば．プログラムが終了するかどうかを前もって知る方法はないからである．したがって，長時間走っているプログラムは正しく実行しているか，あるいは無限ループに入っているかどちらかであろう．この理由のため，バッチ・システムはいつでもプログラムを途中でお流れにする**ある種**の手段を提供しており，そのプログラムがシステムのリソースを不当に独占することを防いでいる．普通，これはマシン・オペレータによってなされる．彼はプログラムがジョブ・カードに規定されているユーザによる推定時間より長く走ると，オペレーティング・システムからメッセージを受けとるのである．同じ型の機構がタイムシェアリング・システムでも利用されうるのであるが，これ**のみ**では不適切であろう．というのは，そのユーザがすぐそばにおり，ユーザ自身，プログラムがあまりにも長く走っていて，どうも誤りらしいと気づきうることが多い(しかし，いつもそうとは限らないが)からである．また，ユーザは，中間結果(これはバッチ・システムのラン中はだれも見れない)のプリント出力を見て，この結論に達するかもしれない．

たとえば，端末の“アテンション”ボタンを用いて，ユーザプログラムを中断することができるようになっていなければならないが，その際ユーザが応答するために必要とする機能はあまり明確なものではない．最も簡単なものは，ユーザに翻訳の再スタートとプログラムのランを要求する（普通，ソース・ステートメントあるいは入力データの修正後)．これはいかなる特別なコンパイラ能力も必要としない．この慣例によれば中断のたびに新たな完全なコンパイルを行なうので，タイムシェアリング・システムは現存のバッチ・システムからほとんど変更なく(少なくとも中断要求に応答するように変更するが)移行したコンパイラを使用する．コンパイル時間を縮小するために（オブジェクトの速度を犠牲にすることが多いが)，Waterloo 大学で非常に成功裡に開発された FORTRAN IV 用の WATFOR のような高速コンパイル・コンパイラが望ましい．もしシステムがユーザに中断されたプログラムを再コンパイルなしに後の時点で再開続行することを許さなければ**ならない**とすると，付加的な機能が提供されなければならない．何が必要とされ，そしてこれらがどのように用意されるかを見るために，ユーザが中断要求に引き続き何を行いたいと望んでいるかを考えてみよう．第1に，おそらくユーザは(ループ制御変数のような)あるデータ値を“見る”ことを望むであろう．もちろん，ユーザはこれを自分のプログラムで使用している名前で参照するより他はない．この名前をそのデ

ータ値のメモリ中のアドレス(番地)に変換するためには，普通コンパイル後には捨てられてしまうシンボル・テーブル(そしておそらく再配置テーブルのような他のテーブル)を必要とする．したがって，この要求を満たすためには，システムはコンパイラによって作られたシンボル・テーブルを保持しなければならない．また，要求されたデータはソース言語の形式でユーザに表示されなければならないが，これはまた少くとも通常のコンパイラ機能のいくつかを必要とする．要約すると，ユーザの要求があり次第それに応じて**計算結果の値**を表示したり修正したりする能力には，バッチ環境のために書かれたコンパイラでは必要がなく普通利用できないような特徴が要求される．

ユーザの構文上のエラーに応答する問題は(図9-1.2の項目6)，タイムシェアリングあるいはバッチのいずれのプログラミング・システムにとっても非常に重要である．タイムシェアリング・システムがここに独自な要求を持っているかどうかは，いくらか議論の的となっている．ある人は，入力中にプログラムの行毎を構文的にチェックし，いかなるエラーもできるだけ早くユーザに送るべきであると(強く)信じている．他のある人は，プログラム全体が入力された後に，すべての誤りを表にした簡潔な報告があれば充分で，しかもその方が望ましいとさえ(強く)感じている．それは個々の人間の作業習慣に依存するともいえるので，この問題は一度にあらゆる場合に対して解決はみられない可能性がある．これを書いている時点(1975)では，IBM の APL や CMS システムのような，もっとも成功しているタイムシェアリング・システムのいくつかは各行毎の構文チェックをしていないが，他のシステムでは行なっている．各行毎の構文チェックの特徴を提供することは，コンパイラを最初から書き始める場合にはたいして困難なことではない．しかし，あるコンパイラをこの特徴を持つように修正するには，決して取るに足らない変更ですむというわけにはいかない．

上の議論を要約すると次のように言えよう．高速コンパイル・コンパイラがあれば，たとえそれがバッチ・システム向きに設計されたものであっても，それを人間が巧みに操れる良好なファイル編集処理サブシステムと共に用いて多くの目的のために適切なインタラクション機能を供給することが可能である．しかしながら，そのようなシステムでは，デバッグのために(必須のものではないとしても)望ましいある種の機能，特に計算結果のデータ(それはソース形式で参照される)を修正したり，中断されたプログラムを続行再開させる能力は提供されない．

トランスレータ・プログラムが**コンパイラ**というよりむしろ**インタプリタ**で

あるならば，上述の望ましいすべての機能(およびその他のいくつか)は，ユーザおよびトランスレータ・プログラマの両者にとって，最も使いやすいものとして提供されよう．インタプリタはソース・プログラムの各ステートメントをそれが**実行される**たびに翻訳するが，一方コンパイラはユーザ・プログラム全体を翻訳フェーズという別個のフェーズの間に機械(オブジェクト)プログラムにいったん翻訳し，その後でその結果のオブジェクト・コードを実行する，ということを思い出そう．普通，インタプリタはその時間の大部分を，生産的な実行に費やすというよりむしろステートメントの**繰返し翻訳**に費やすのである．**図 9-1.3** はコンパイラ対インタプリタの相対的な性能特性の概略図である．これによると非常に短いランのプログラム（特にループのないもの）では，（コンパイラの場合にはその初期作業があるので)インタプリタの方が速いということがわかる．一方，ループのあるプログラムではほとんどの場合コンパイラの方が有利であることがわかる．しかしながら，インタプリタの場合は各ステートメントをそれが実行されるごとに翻訳するので,再コンパイルの必要なく，プログラム内の**変更**を容易に行ないうる．また，インタプリタは記憶装置割当とリソース使用については“最後の瞬間”の情報に頼っているので，その言語はいくらかより使いやすいものとなりうるが，一方コンパイラの場合にはこれらのことについてプログラムが走る**以前に**(翻訳されている時に)ユーザの宣言や規則によって知らねばならない．別の言い方で言うと，インタプリタは，ユ

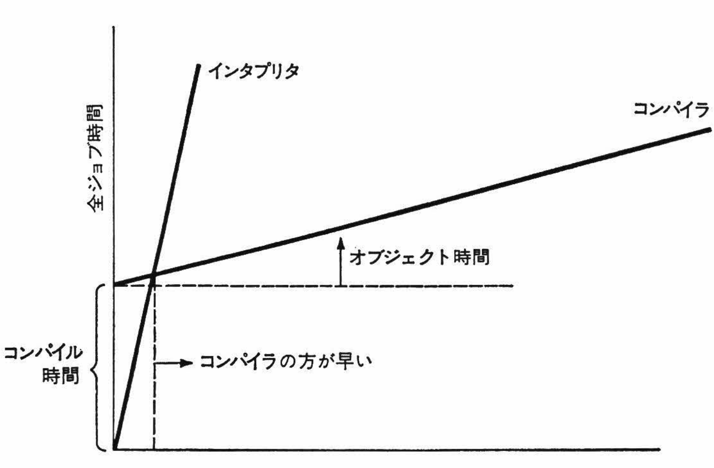


図 9-1.3　コンパイラとインタプリタの性能対比を考察するための概略図

ーザ向きの名前とパラメータをマシン・リソースにできるだけ最後に結びつけるバインディング(binding)タイムを必要とする．実際には，コンパイラとインタプリタの間の区別は上述の議論で見られるような――そこでは説明のために差異を強調したが――厳密なものである必要はない．それにもかかわらず，コンパイラがインタプリタ技法を利用すればするほど，ユーザにとっては便利になるがランの時間は悪くなってくるのである．

プログラムとデータ操作の編集を含む基本的なマン-マシン・インタラクションは普通非常に頻繁に行なわれるが，それには普通ほとんど計算時間を必要としないということは，いくら強調してもし過ぎるということはない．この理由のために，それらは**軽微**(trivial)**トランザクション**と呼ばれよう．ある意味で，これは不適当な用語である．というのは，それらは急いでサービスを受けるべきという重要性においてもっとも高い地位を占めているからである．言い換えれば，軽微要求に対して速く応答することは可能であるというのみならず基本的に欠くことができないことでもある．

軽微トランザクションに対する迅速な応答の必要性は，マン-マシン心理学のより一般的な原理の特例として考察しよう．

〈応答の設計原理〉

人間のユーザが迅速に処理されることを期待している要求は，もしそのようなおのおのの要求を単独で走らしたとき短時間で処理できるなら，それに対してもっとも速い応答を与えるように，システムを設計すべきである．

この原理は，一見して，むしろ明白なものであるが，もしそれが受け入れられたなら，リソース・スケジューラの設計にとって非常に有用な指針を与えることができる．したがって，プログラム編集・表示の軽微なトランザクションに対してもっとも高い優先度を与えるのみならず，その他の短時間トランザクションにもまた，より長時間のランのものよりも高い優先度を与えるべきである．ラン時間が前もってわかっていないことが多いので，ラウンドロビン(4章参照)のようなスケジューラは短いトランザクションを，たとえそれらを識別できない場合でも，迅速に完結することができる．また，ユーザはあるトランザクションのサービスに要する時間として，システムがそのトランザクションを処理する時間より，ずっと短いものを期待することがよくあることは言及に値する．その場合その応答は**必然的に**不満足なものとなるであろう．しかしながら上記の原理は，このような矛盾がない場合にはシステムは人間の期待に

答えるように応答すべきであることを強調しているのである．

“ランの長さ”(およびあるシステムでは必要とされる記憶容量)の期待値を知っていると，それはスケジューラにとって非常に価値のある助けとなりうる．ラウンドロビン・スケジューラはそのような知識を利用しないが，このスケジューラはその簡単な形態ではかなりのスワッピング遅れを要することがあり，この遅れが，そのスケジューリングの利点のいくつかをだいなしにする傾向がある．トランザクションの処理時間を前もって知ることができるかどうかの問題は奥深い問題である．ただ1つの(あるいは，ほんの2，3の)特定の言語とファイル編集プロセサしか持たない専用システムでは，設計者とスケジューラは重要な種類のトランザクションのソリース要求についての情報を容易に得ることができる．たとえば，1つのFORTRAN言語コンパイラが使用されているIBM RAXシステムは，スワッピングなしで完全なコンパイルを行なうように設計することによって，スワップ・オーバヘッドを減らした．というのは，ほとんどのコンパイル時間は軽微要求に不当に影響しないような充分短いものと判断されたからである．しかしながら，RAXはオブジェクト・プログラムの実行においては，そのラン時間は予測できないので，“タイム・スライス”(およびスワップ)を行なった．

言語トランスレータ，ファイル編集機能などの多様性の点で，システムが一般性を持てば持つほど，リソースの使用についての前もっての知識を利用することが困難になってくる．したがって，一般的なシステムは，リソース・スケジューリングのために，リソースの使用状況についてのそれ自身の観察と測定にますます多く頼らねばならない．

9-2 タイム・スライスの選択 [BE, COR, LA, IB2参照]

タイム・スライシングの基本的な目的は，オペレーティング・システムにそのリソース・スケジューリングの決定を再考するように強制することである．このことは，良好な応答性能についてのほとんどの尺度にとって本質となっている．というのは，スケジューリング変数(4章参照)，および優先度は時間の進行にともなってそれらの値が変化してゆくことが多いからである．これらの変数およびそれらの相対的順位の計算にはユーザにサービスしているのと同一の中央計算機の使用が必要となるので，これらの計算を行ないそれに基づきリソースの割当の決定を行なうために，時々(タイム・スライス間隔)CPUを割込使用するのは基本的である．

スケジューリング変数の更新とジョブの切換に要するオーバヘッドを無視し

て，最も単純化した観点からながめると，スケジューリング変数の状態の変化に対する高速な応答を保証するためにタイム・スライスは非常に短くあるべきである．しかしながら，より現実的には，1つのトランザクションから他のそれへのCPUの切換には2つの切換がともなう．すなわち(1)現在のユーザ・プログラムからオペレーティング・システム（スケジューラ）へ，そして(2)オペレーティング・システムからユーザ・プログラムへ，である．たいていのシステムでは，第1の段階はかなり速い．第2の段階もまた，もしその復帰が段階1の話に出てきたプログラムへの復帰ならば，速くできる．しかしながら，もし段階2に新しいプログラムがかかわりあって，もしこのプログラムが低速記憶装置から主記憶装置へスワップインしなければならなく，さらにもしこれに先立ってすでに主記憶装置にあるプログラムをスワップアウトしなければならないとなると，ジョブ切換の過程はまったく時間を消費するものとなってしまう．それゆえ，ジョブ切換のオーバヘッドは実現可能なタイム・スライス値の下限に当然影響するであろう．しかしながら，多くのシステムには，システムが切換わることができるジョブには2組ある．すなわち，主記憶装置内に現在駐在しているものと，現在駐在していないものの2つである．第1の組の切換オーバヘッドは，第2のそれよりはるかに低いが，このことから**2種のタイム・スライス**があってもよいことがわかる．いくつかのシステムは可変のタイム・スライスを持っているが，そのような場合，もし最大タイム・スライスが長い(たとえば数秒)ときには，CPUは新しいトランザクションにとって割込可能となる．というのは，それらのトランザクションは重要な軽微応答のカテゴリに入るかもしれないからである．

要約すると，タイム・スライス値があまりに大きいとシステムの応答を悪くし，一方，あまりにも小さいとCPUのタスク切換オーバヘッドを増加させる傾向がある．もう1つの主な非能率の源であるスワッピングを減少させ，さらに出来るだけ応答感度を保つように努めるために，次の2つの時間に対して別々のタイム・スライス間隔が選択されることが多い．すなわち(1)あるトランザクションがスワップアウトされる前に，主記憶装置に残っていなければならない時間，および(2)主記憶装置内の別のトランザクションに切換えられる以前に，あるトランザクションにCPUが割付けられている時間，である．タイム・スライスの長さは，システム設計時に固定されてもよいし，あるいはシステム・プログラマあるいはオペレータによって供給されるパラメータとして規定されてもよいし，また，ジョブとシステム活動状況についてのシステム自身の観察に基づいて自動的にオペレーティング・システムによって計算されて

もよい．特定の場合についての例が後に与えられよう．

9-3　MIT CTSSシステム［COR, SC 1 参照］

1960年代初期にMITで開発されたCompatible Time-Sharing System (**CTSS**)は文献に詳細に報告された最初のシステムの1つである．その名が示すように，通常のIBM 7090コンピュータ上で走るいかなるプログラムも変更することなく(互換性を有して)タイムシェアード・システム上で走るように，このシステムは設計された．この要求(これは望ましいことではあるが，タイムシェアリング・システムにとって本来必須のものではない)は，ユーザがタイムシェアリング・システム上でプログラムを作り，デバッグし，修正し，そしてより機械効率のよいバッチ・システムで生産的ジョブとしてこれが走ることを確かめるのに都合がよい．これはまたバッチ・ユーザが利用できるすべてのリソース，特にすべての主記憶装置が，またタイムシェアリング下でも利用できることを意味している．

CTSS装置構成の概略図は，**図9-3.1**に示されている．2つの32k語-2マイクロ秒コア記憶装置ボックスをもった単一IBM 7094中央処理ユニットから

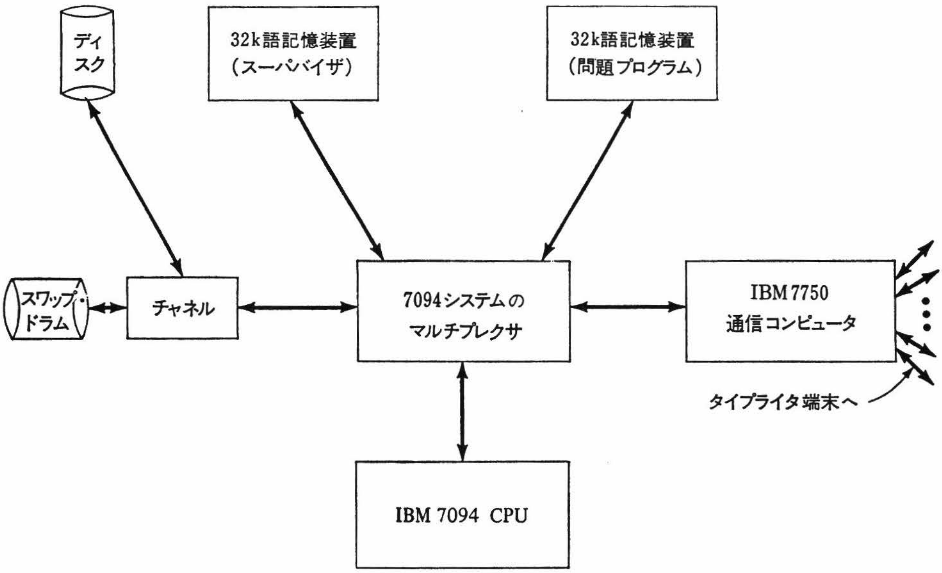


図 9-3.1　MIT CTSSシステム構成(一部)

なっていることがわかる．これらのコアの1つは，一時にただ1つの問題のプログラムのみに使用されたが，それゆえ互換性の要求が満たされた．というのは標準の7094システムには32k語の記憶装置しか装備できなかったからである．他の32k語コア・ボックスはCTSSのシステム・プログラムのために使用された．ほとんど他のIBM 7090/7094システムと同様に，このシステムにもまた回線多重化・制御のために特殊目的型IBM 7750計算機が用いられている．高速チャネルが主コア・ボックスとドラム・ディスクの間の転送を制御していた．ドラムは主要な“スワップ”装置であった．

ユーザが利用できる最大可能記憶空間を保証したこの互換性の特徴は，リソースをより柔軟な方法で管理するシステムで起こる多くのスケジューリング・制御問題を除去した．したがって，CTSSはコア記憶装置には一時にただ1つのユーザ・プログラムしか保持しないので，空間の共有はなく動的プログラム再配置の必要はなかった．CTSSによってもともと使われたIBM 7090コンピュータは特別設計としてこれらの特徴を持っていたが，それらは使用されなかった．主記憶装置は一時にただ1つのプログラムしか保持しないということのもう1つの結果として，1人のユーザのジョブを別のジョブのCPU作業と“オーバラップ・スワップ”させることができない．(オーバラップ・スワッピングの議論についての9-6節を参照)．ほとんどのシステムに当てはまることだが，スワップ時間を縮小するために，CTSSは最大32k語の空間のうち使用中の部分のみをスワップした．

後のタイムシェアリング・システムに共通なスケジューリング問題の多くはCTSSにはなかったが，このシステムは1つの主要なスケジューリング機能，すなわち，次にどのトランザクションが，どのくらいの時間，サービスを受けるべきかを決定すること，をエレガントでしかも興味ある方法で行なっていた．**図9-3.2**はCTSSの“指数”スケジューリング・アルゴリズムを要約したもので，次にこれについて少し述べる．スケジューラは，**新たに**到着した要求に高い優先度を与える．というのは，それは多分高速応答がもっとも重要な短いインタラクティブな要求かもしれないからである．その場合にはそのトランザクションはすぐにサービスされるであろう．トランザクションがより長い処理時間を要求することがわかるにつれて，その優先度はより低くなってゆく．しかしながら，トランザクションの優先度が落ちるにつれて，ということはサービスのためにそれが選ばれることがより少なくなることを意味しているが，一度選ばれるとそのサービス期間(タイム・スライス)は高い優先度のトランザクションに対するものより長い．この後者の特徴は，長いCPU時間を必要と

1. 9優先度レベル；多くのジョブが各レベルに存在しうる．

最高優先度	0	
	1	
	2	新ジョブ：空間 < 4095 語
	3	新ジョブ：空間 ≥ 4095 語
	4	
	5	
	6	
	7	
最低優先度	8	

2. ジョブ・スケジューリング・アルゴリズム
   (a) 次の CPU サービスを受けるトランザクションは，最も優先度の高い空でないレベルから選ばれる．
   (b) 60 秒間選択されなかったトランザクションは，次のより高い優先度レベルに移される．
   (c) レベル q のトランザクションは，サービスのために一度選ばれると，最初の I/O 要求の発生か，ジョブの完了か，あるいは 0.5×2^q 秒経過するまで走る．最長可能バースト・タイム = 128 秒．
   (d) もしトランザクションがそのバーストの終りで完了しないと，それは1優先度レベル下に移される．
   (e) 現在実行中(バースト)のトランザクションは，レベル q に新たな到着があると，現在のジョブがすでに少なくとも 0.5×2^q 秒間走っているならば，それに優先されてしまう(スワップ・アウトされる)．被優先ジョブはその現在のバーストの開始直前の待ちレベルに復帰される．

図 9-3.2　CTSS 指数スケジューラ

するジョブに対するスワッピング(その時は CPU アイドルである)を減らし，したがってスループットを改善することを意図している．要約すると，CTSS は新しいジョブは短かいと仮定し，それには最初良いサービスを提供する．ジョブがぐずぐずしていると，長いものということがわかって，それは“バックグラウンド”にすべり落ちてゆき，あたかもバッチ・ジョブであるかのように扱われた．上述の原理の1つの精密化が行なわれたが，それは，必要とする空間と(未知の)ラン時間の相関を仮定することによって，到来ジョブのジョブ長の予想順位を評価するというあらっぽいものであった．こうして，4095 語以上を要求するジョブは，それ以下の空間を要求するジョブより1優先度低く初期セットされた．

9-4 APL システム [AN, BRE, HE1, HE 3, IV 参照]

APL システムの名は K. E. Iverson の著書(A Programming Language)の名にちなんで付けられた．この著書では，1つの有能でエレガントなプログラミング言語が記述され，広汎な種類のアルゴリズムに適用されている．その言語は，最初の機械翻訳が現われる以前の少なくとも6年間は，小さいがしかし献身的な人々の集団によって“紙上で”徹底的に用いられた．したがって，ほとんどの他のプログラミング言語と違って，この言語は，インプリメンティションによって凍結されたり，便宜と互換性に対するそのインプリメンティションの避けがたい圧力によって凍結されたりする前に，長期間の注意深い概念的な発展を享受してきた．

明らさまにタイムシェアリングを目的として設計されたものではないが，Iverson 言語はマン-マシン環境に理想的に適合している．これは，ユーザに最小のシンボルしたがって最小のキイストロークで多量の処理を指定することを許しているのみならずそれを助長してさえいる，その言語の考え方によっている．言語の次の特徴はこのことを明らかにするのに役立つであろう．

1. 1つの名前で1つの配列，すなわちベクトル，行列，あるいは多次元構造を示すことができる．配列では，ユーザによる特別な注意なしに，そのサイズを縮小したり拡張したりすることが許されている．
2. 多種類のオペレータは，おのおのそれ自身固有のシンボル（あるいは簡単なシンボルの組合せ）によって示されるが，これらは通常の単一変数あるいは添字付変数のみならず配列全体に適用できる．これらは，その他の多くのものと同様に，行列代数のオペレータの意味深長な一般化を含んでいる．
3. 3つのデータの型(数，ビットおよび文字)には，ユーザが明らさまにはまったく何もしなくても，ビットと数の型の間（そしてまた整数と浮動小数点数の間）の自動的な翻訳が用意されている．

これらの多くの性質のマン-マシン・インタラクションに対する有用性が，APL の言語の概念の多くを用いた“パイロット”システムによって1964年に実演された．

この言語は多年の予備的インプリメンティションを通じて発展し，そのインプリメンティションの間にさらに一層改善された．たとえば，スキャニング・ステートメントに対する厳格な**右左則** (right-to-left rule) が最終的に洗練さ

れたものとなり，また多次元配列への拡張が完成された．APL のインプリメンティションは Yorktown, N. Y. にある IBM の Watson Research Center において K. E. Iverson および A. D, Falkoff の総監督の下でなされた．

IBM システム/360 のためのトランスレータは(コンパイラに対立するものとしての)**インタプリタ**であり，それは1年未満の期間で主として L. Breed および R. Lathwell の2人によって作られた．このペアの若い専門家は，また，I. M. Sharp Associates (Toronto, Canada) の R. Moore の助力によって，ユーザに限られた量のディスク空間のアクセスを許す簡単なコマンド言語のみならず，端末操作およびスワッピングのためのソフトウェアを作成した．初期の APL システムは，DOS，すなわち1960年代中期に中小型システム/360 で用いられた主な IBM オペレーティング・システム，の下で走った．その後(1968年頃)，APL はより包括的な OS/360 オペレーティング・システムの下で利用可能となった．250 k バイトぐらいの小さな主記憶装置を持つモデル 40 およびモデル 50 システムの多くが約 50 の接続中の APL ユーザにサービスしたが，このシステムは少なくとも 500 k バイトを持つモデル 50(あるいはより高速な CPU)を含むのがより典型的である．これらのシステムでは，約 50 以上の端末に APL の軽微要求に対して通常3秒という快適な範囲の時間で応答するのみならず，OS バッチ作業負荷を走らすことができる．

性能の観点からは，APL は2つの面から考慮されよう．第1に，言語とそのトランスレータがある．これらはタイムシェアリング環境を無視して分析しうる．第2の面はスワッピングのような多重ユーザのリソース割当問題に関連している．

言語のエレガンスさとユーザの使い勝手さを出来るだけ妥協させないですむようにと，**インタプリタ**・トランスレータを使用することに決まった．言語をコンパイラによってインプリメントする場合には，いずれも宣言やおそらくその他ユーザがうるさがることを必要とすることは，ほとんど確かである．しかしながら，インタプリタはあらゆるステートメントをそれが実行されるたびごとに翻訳しなければならない(コンパイラに関していえば，プログラムにつき一度でよいのだが)ので，実行時におけるインタプリタの価格はかなりなものとなる．しかしながら，このインタプリタの大きな非能率は(他の言語に比較して)いくらかやわらげられる．というのは APL 言語は強力な配列オペレータのおかげで，結果的にステートメントあたりの実行の量が多くなる傾向にあるからである．しかし，かなりループがあるプログラムにおいては性能は悪くなる傾向にある．APL のインタプリタ法による翻訳の**空間効率**は，コンパイ

ラのそれに比べてより良好と思われる．APL はインタプリタの単一の写しを常駐型で用いており，すべてのユーザはこれを共有している．これは，高速応答を目的として設計された常駐型コンパイラによって占められる記憶容量に匹敵する空間を要求する．しかしながら，現実に空間の節約となるのは，この言語によると他の言語では大きなプログラムになるものが簡潔なプログラムですむという事実のおかげによる．インタプリタは，一時に1つのステートメントのための中間スクラッチ記憶装置しか必要としない．一方，コンパイラは，より大きなオブジェクト・プログラムのための記憶装置を必要とする．

APL の時間・空間利用の間には興味ある相互作用がある．配列をうまく利用し，その結果非常にコンパクトなステートメントとなる多くのオペレーションは，それ自身はほとんど記憶容量を必要とせず，また翻訳するのも速いが，不幸なことに，かなりの記憶空間を必要とする．非常に簡単な例として，最初の N 個の整数の和をもっとも簡単でもっとも高速な APL 流のステートメントで表わすと，

$$+/\iota N$$

となる．しかしながら，このステートメントは APL に N 個の数を記憶するように要求するが，これは同じ計算をループでプログラムした場合に要求されるよりもはるかに(大きな N に対して)多い．しかしループはプログラムにとってはより不便なものであるし，ランもずっと遅くなるであろう(APL では).

これらの考察が，APL トランスレータのパフォーマンスについてあらゆる種類の事柄(有利なこともそうでないことも)を“立証する”のに，容易に利用できる．インタプリタ・トランスレータを用いることによって，ユーザは言語の使いやすさと引きかえにかなりな時間の代価を支払っているが，この代価は，多くの配列オペレータを用いてそれにより空間をひんぱんに惜しみなく使うことによって減らし得るのである，と述べることは公正なことと思われる．もちろん，空間効率はその人の空間が使い尽されたときのみ実際に重要となる。それまでは，そして，これが小さな問題に適用されている限りは，(出来るだけループの少ない)もっとも簡潔なプログラムがもっとも便利でしかも適度に速い．

APL の1つの興味ある重要な性能の特徴は，“I ビーム”オペレータのセットである(それらを呼ぶオペレータが，一見鋼鉄ビームの横断面のように見える2つのシンボルの合成であるので，そう呼ばれる)．どの端末でも使えて，**いかなるプログラムにも含まれうる**これらオペレータの一部のリストが**図 9-4.1**

X	*IX* の定義
19	このセッション中の累積打鍵時間[1](キーボード・アンロックの合計時間)
20	時　刻
21	このセッションの CPU 使用時間
22	作業空間内の残存バイト数
23	現在使用開始中の端末数
24	セッション開始時刻
25	日　付[2]
26	*I*27 の最初の要素
27	状態ベクトル内のステートメント番号のベクトル

† **注意**: 1. すべての時間は 1/60 秒単位である.
2. 日付は 6 桁整数で表わされる. 2 桁で月, 日, 年を表わす.

図 9-4.1 いくつかの APL I ビーム.

に与えられている. I ビームのうちのいくつかは, 個々のプログラマが自分のプログラムの性能を評価し改善するのに, 大いに価値がありうる. これらにはプログラマの作業空間に残っている記憶容量を与える I22 や, プログラマ各自の CPU クロックの読みを与える I21 が含まれる。後者のものを使用するとプログラムのいずれの部分でもそれを実行するための CPU 時間が 1 つの APL 変数として得られる. プログラムの CPU 時間は, タイムを取られているプログラムの性能にのみ依存し, 多数のユーザによるシステムの共有には関係していないことに注意しよう. その他の I ビーム(14 以下の番号を付けられた)は, 種々の型のシステム活動の頻度分布のベクトルを参照する. これらの I ビームは, 大切なコアメモリ空間を保護するために, “特権を与えられた”端末のみに制限されるかもしれない. 要するに, I ビームによって, いかなるユーザも個々のユーザとシステムの性能の貴重な様相にアクセスすることができる. (システム性能の研究におけるこれらのより多くの例については, 9-6 節を見よ.)

さてここで APL 性能の第 2 の面, すなわち多重ユーザの処理について取りかかろう. **図 9-4.2** は主記憶装置の概略図を示す. 主記憶装置内に小記憶領域(約 600 バイト)が, 使用開始中のおのおののユーザごとに, 端末メッセージ・バッファとして確保される. この空間と端末の間の伝送は, CPU とは独立に動作することができる, マシンのマルチプレクサ・チャネルによって制御されるので, 多くのそのような伝送は互いに並行して, そしてまた CPU 処理と並行して行なうことができる. したがって, たとえユーザの作業空間がスワッ

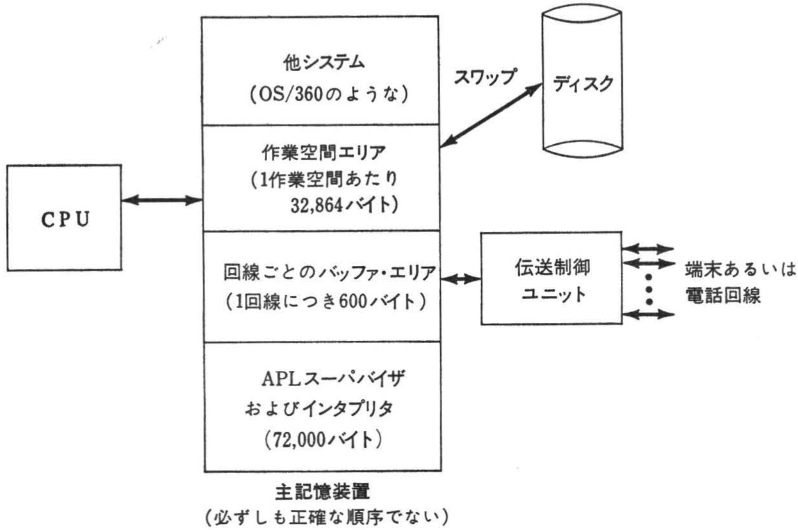


図 9-4.2 典型的な APL リソース

プ・アウトされてしまっていても，少なくともある限られた時間で，ユーザはプログラムやデータを入力したり，出力を受信したりすることができる，ということになる．システムはまた，CPU 時間と端末伝送とのオーバラップが(異なるプログラムの間のみならず)同一のプログラムに対してさえも可能となるように，設計される．したがって，“1 行の計算/1 行のタイプ”の順序を繰返すプログラムは，もし計算部分が過度なものでなく，また CPU サービスに対する要求の総数がそれほど多くなければ，端末にとってはプリント出力の速度で走っているようにみえる．

主記憶内の APL トランスレータの単一の写しはすべてのユーザによって共有される．導入にあたっては，2 つまたはそれ以上の作業空間(各 32 k バイト)を同時に主記憶装置に持つことが指定されるであろう．スワッピングはこれらの領域とディスク (IBM 2311 あるいは 2314/3330) の間で行なわれる．2 つ以上の作業空間が主記憶装置に同時に保持されるので，(異なる APL ジョブでの)入力・出力スワッピングと計算の間でオーバラップが可能である．この潜在的な性能上の利点についてのある量的な考え方が後に (9-6 節参照)議論されるであろう．IBM 供給の APL システム(1972 年現在の)の制限の 1 つは，ユーザに対する作業空間が固定サイズ (通常 32 k バイト) であるという制約である．ただ 1 つの作業空間しか“活動中”とならない，すなわちユーザ・プロ

グラムによって一時にアクセスできるのはただ1つのみである．非活動中の作業空間をアクセスするには手入力のコマンドが必要とされる．この重大な実用上の制限は少くとも1つの(Scientific Time-sharing 社によって提供されている)APL システムでは取除かれている．システム/370 のために1973年に発表された APL の SV(shared variable)版は，ファイル・アクセス能力をもまた提供できる．APL は最小機器構成のどの標準 IBM S/360 あるいは S/370 上でも走るということは言及に値することである．いかなる種類の特別の（非標準な)装置も必要ない．

異常に高い信頼度が APL 運転のはじめの数日のうちにも明らかとなった．これは，ある部分は開発とデバッグの間設計者とインプリメンタによってシステムが使われていたからである．APL システムは，典型的には，休日をはさむ数日間も“ほったらかして”(機械室オペレータなしで)走るのである．

9-5　性能測定 [AN, BA, DO, HE 3, MA, SC 1, SH 参照]

性能測定技法のいくつかの特性を，ハードウェア・モニタについては第4章で，ソフトウェア測定手段については7-9節で議論した．この節では，タイムシェアリング環境，すなわち人間が配置された多くの端末にサービスする計算機システムに適用できる手段に注意をしぼろう．そのようなシステムは，もちろん，またバッチ作業負荷を走らせるかもしれない（そしてそうしていることが多い)．測定技法には，内部的および外部的の，2つの基本的なクラスがあり，これらおのおのは，測定を行なう人に必要とされる知識や手腕が非常に異なるものであることを意味する，まったく異なる型の測定実験となる．

内部的手法では，測定に関係するものはオペレーティング・システム・プログラムにアクセスできなければならないし，または，それを修正すらしなければならないかもしれない(現代のほとんどのオペレーティング・システムには，多くの測定機能がもはや含まれているとはいいながら)．バッチ・システムと同様タイムシェアリング・システムは，割込解析プログラムを持っていることが多いが，これによってすべての割込を途中で捕らえ，そのおのおのに対して後のデータ整理のための時刻刻印と事象の記述を与える記録を取るのである．タイムシェアリング・システムでは，それに加えて，何らかの方法で端末ユーザのおのおのの活動を捕える必要がある．このデータは通常バッチ・システムよりはるかに多いので，測定のからくりが性能を無視できぬほど下げてしまうことがないように，一層注意が必要である．文献に報告されているタイムシェアリング・システムのほとんどではないにしても，その多くが，いくつか

の内部的測定機構について述べている．しかしながら，**どの**内部的ソフトウェア測定手段も，**たった1つの**オペレーティング・システムに特定なものであるということはほぼ確かであることに注意しよう．それは開発の段階では，不可欠なものでないにしても，有用な手段であり，またシステムを導入する人にとって大いに価値がありうるけれども，内部的測定の利用者は通常システムの内部構造について完全に知っていなければならない．このことは内部的手法に本来備わった性質であるという必要はない．注意をはらえば，この方法で得られた性能データはシステムの内部の知識を必要としない形で表示することが可能なのである．しかし少なくとも現在は，これは事実というより期待を言っている．

測定の**外部的**手法は，システムを一種の“ブラック・ボックス”として取扱い，通常のユーザ端末のみがすべての測定を行なうために利用できる．内部的手法と異なり，外部的機構は，作業負荷の制御およびシステムがユーザにどのように見えるかということに，決定的に依存しているが，システムの内部的振舞の詳細についての知識やオペレーティング・システムの内部への立ち入りはまったく必要ない．

4章で議論したように，タイムシェアリング・システムを含む，いかなるシステムの性能も3種の要因に依存している．

1. 作業負荷
2. システム構造(ハードウェアおよびソフトウェア)
3. 性能尺度

われわれは，いま性能の**測定**について考察しているのであるから，もちろん，実際のシステムがわれわれに提示されていなければならない．それゆえシステム構造は完全に決定されており，報告書の中にそれについての注意深い記述を含める仕事が残されている．項目3，すなわち性能尺度もまた手におえない問題と思う必要はない．特に，次の基準が代表的である．

1. **応答時間**
   (a) 平均，中位数，標準偏差
   (b) 頻度分布(実行時間を値域しとて)
2. **中央システム利用率**
   作業負荷の実行中における，CPU，チャネル・スワップ装置などのような各要素のビジー時間の百分率のベクトル

残りの項目の**作業負荷**は測定のために固定させるのが最も困難なものである．この節のほとんどがこの問題に集中しているのはこの理由による．その理由は容易に理解できる．タイムシェアリング・システムにおける作業負荷はシステムと直接インタラクトしている**人々**によって与えられるので，端末における人々の振舞を作業負荷の仕様の一部としなければならない．ただちにわかる1つの結論は，行動科学者にとっては珍しくないが，物理科学者を悩みかき乱させるものである．生のユーザの活動を使うことを主張するならば，**再現性のある**結果を得ることはきわめて困難であろう．しかしながら，よく注意すれば，その結果は**統計的**に再現可能であるかもしれない．たとえば，連続する就業日の同じ時刻に同一のシステム上で測定するならば，各ユーザの活動の詳細はほとんど間違いなく異なるであろうが，それらの集合的な活動の頻度分布は多分似ているであろう．いうまでもなく，これは確かめられた事実としていっているのではなく，むしろもっともらしいというだけのことであって，それ自身測定してチェックされるべきことである．実際のシステムの活動を監視することは，内部的測定手段で容易に可能である．外部的測定技法を用いることはずっと困難であるが，ある種の性能情報は**スヌーパ**(snooper：うろうろのぞき回る人)端末を利用して外部的に得ることができる．スヌーパ端末(1台または複数台)は普通のユーザ端末から成っていて，その端末で測定ユーザは，ひと続きのインタラクティブなサービス要求，すなわち選択されたコンパイル後実行ジョブのような，既知の測定可能な仕事を要求する．この間，他のユーザはいつもと同じように振舞っている．スヌーパは，自分の要求に対するシステムの応答性を測定し，それが(たとえば)時刻やスヌーパの要求の性質によってどのように変るかを記録する．スヌーパはまたこれらの時間を(なるべくなら同一のスヌーパ作業負荷で)自分がのぞき回る別のシステムのそれと比較することを望むかもしれない．

タイムシェアリング・システムに対する作業負荷の特性描写の最も基本的な概念は**スクリプト**(script：台本)である．スクリプトは実ユーザあるいは想像上のユーザとシステムのインタラクションの記録の収集から成っており，おのおのの記録上の各項目には時刻が刻印されている．1人のユーザに対するスクリプトを思い描くために，ある端末でのセッションを想像しよう．この端末では注釈付きの端末印字出力が行われるので，各行には，その行が出力あるいは入力として現われた時刻が刻印される．測定実験のためのスクリプトはそのような個々のユーザのスクリプトの集合から成っている．そのような収集が得られ，それが適合したものであることに同意がえられれば，**その**スクリプトに対

するシステムの応答を再現することは可能である．その同じスクリプトは機器構成やオペレーティング・システムの変更を研究するための作業負荷記述としても利用することができる．その場合，そのスクリプトはそのシステムのシミュレータに対する入力データとなる．

スクリプトを選ぶことは，ベンチマーク(4-2節参照)の選択の問題と類似の問題である．ここではこの問題を議論**しない**で，単にあるスクリプトが利用できると仮定するにとどめよう．スクリプトの概念は原理的にはまったく簡単であるが，たとえ受け入れることができるスクリプトが存在している場合でさえ，それを表現し適用する機構は手に負えないものになる可能性がある．スクリプトの情報量は大きなものになりうるうえ，その時間順序がすべてを決定するので，最も単純なものを除いてはいずれのスクリプトもそのマニュアル入力は実現できない．すぐ後でスクリプトの機械表現を考察しよう．しかしながら，次に示すように，**非常に簡単な**スクリプトを**マニュアルで**システムへ適用することはありえることで，経済的で，実現可能であるということを手短かに示すためにわき道にそれることは価値がある．

1. **予備テスト準備**
   (a) いくつかのテスト・プログラムを選び，デバッグする．そしてそれらを多くのテスト・ユーザのファイルに格納する．1つのスクリプトは典型的には1つのテスト・プログラムを要求する人のそれである(下記を見よ)．
   (b) テストのためにそのシステムを“譲り受ける”時間の手はずを整える．
   (c) テストの時に，そのスクリプトを物理的に入力する人のチームの手はずを整える．それらの人々はテスト下のそのシステムに(ユーザとして)ある程度慣れているべきである．
2. **測定テスト(いくつかの典型的なもの)**
   (a) 軽微応答テスト：同時にj人のユーザにできるだけ急いで軽微要求をさせ，その応答時間をストップ・ウォッチで測定する．最初は$j=1$，それから2，3等とする．
   (b) ラン応答テスト：同時にj人のユーザに同一のプログラムを呼ばさせ，応答時間を記録する．$j=1, 2, 3$などと，テストを繰返す．
   (c) 軽微応答とラン応答テストの混合．

上述の種類のテストは原理的に簡単であるが，その実際の実行はかなりな人

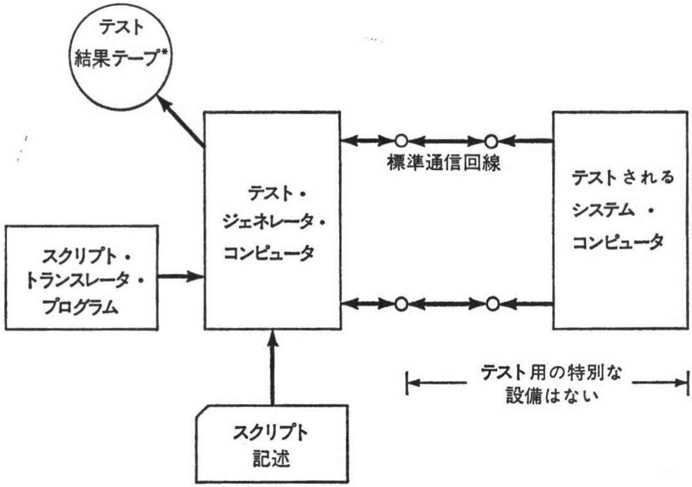
間の協調を必要とする．しかしながら，それにはいかなる特別の装置も必要なく，システム自身をみだりにいじくることもない．テスト・プログラムを思慮深く選ぶことによって，この手法は多くの性能特性と難点を急速に明らかにしうる．

計算機が複雑なスクリプトをテスト下のシステムに供給できるように，それらを機械読取可能な形で表現することは，必要なことでありまた実現可能でもある．スクリプト情報をサービス要求の形に変換し，かつシステム応答を記録するための手続きは，そのスクリプトを入力データとする1つの**プログラム**としてインプリメントされてもよい．テスト下のシステムが**この**プログラムを走らすことさえ可能である．その場合には，システムのリソースはテストされているスクリプトにより共有されているのみならず，テスト機構それ自身によっても共有されている！この場合には次のような明らかないくつかの問題がある．

1. テスト下のシステムのリソース(特に主記憶装置)はテストのために減少させられる．
2. テストの手続には，テスト(される)システムの時間を必要とする．
3. スクリプト出力を与えるべき端末が手に入らないかもしれない．
4. 計算機がテスト・ジェネレーター-レコーダ・プログラムとテスト・システムの間で共有されるためには，ある種のソフトウェア・インタフェースが必要であろう．

最初の2つと最後の点は明らかである．第3の点はいくらかコメントする価値がある．実際のシステムでは，ユーザの要求は典型的には電話回線を通じて，計算機に接続された伝送制御マルチプレクサ装置に到着する．ほとんどのシステムでは，同一のコンピュータがこれらの端末への入力を発生することは**不可能**である！ついでに言えば，そのような特徴は性能測定のみならずシステム・テストにとって有用であるが，多くのシステムは今までのところそれを有していない．もちろん，測定は自己発生型のテストを用いて行なうことは今でも可能であるが，現状では発生された要求は，主記憶装置の端末バッファ領域のようなあるシステム内部領域に，それら領域に適合する形式で供給されなければならない．

ハードウェアに一層金をかけて自己テストの困難を克服しようという機構として，もっとすっきりしたやり方は，テストの供給とその結果の受信と記録に**別の計算機**を用いる方法である．そのようなシステムの概略図を**図 9-5.1**に示



*テスト結果テープは，後に処理される（テスト・ジェネレータ・コンピュータが スクリプトにできるだけ早く応答するため）

図 9-5.1 計算機をテストするために別の計算機を使用する

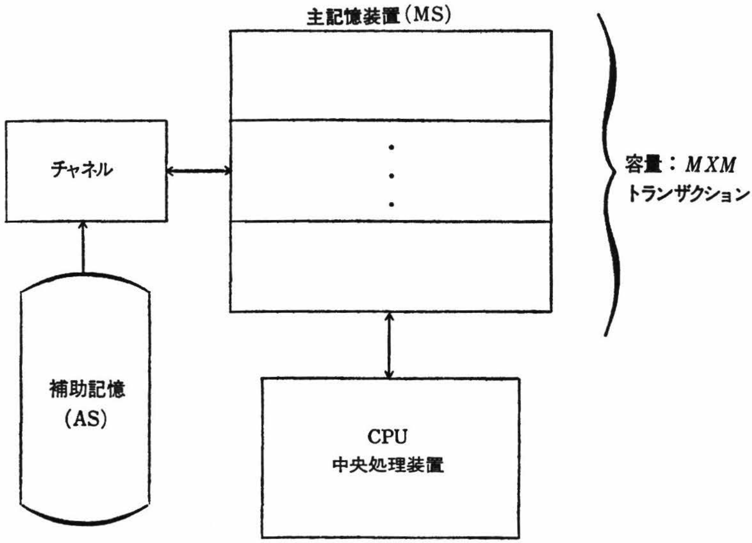
す．テスト・ジェネレータ計算機には，最初，スクリプトを機械形式に変換しテスト・システムの応答を受信し記録するためのプログラムをロードする．それからスクリプトそれ自身が(たとえばカードあるいはテープ形式で)テスト・ジェネレータ計算機に送られる．テスト・システムは通常どおりに運転開始を行なうが，人間のユーザはだれも“使用開始”が許可されないように注意が払われるべきである．テスト・ジェネレータ・システムがその最初のメッセージを送るやいなや，スクリプトはすべて使用開始するので，テスト・システムは応答しはじめるであろう．テスト・システムにおける仮想の観測者は(テスト・ジェネレータ計算機というよりむしろ)電話回線の他端に本物の人間がいないということを知る手段をもっていないということに注意しよう！

9-6 タイムシェアリング・システム・シミュレータ［HE 2, HE 3, NI 1, SH, STR 参照］

さて，スワップ用作業空間を持つ種類のタイムシェアリング・システムの簡単なシミュレーション・モデルを述べ，それを使って性能のある問題点を研究するための具体的な機構を提供しよう．そのモデルは基本的には4章の一般型ジョブ処理モデルの1つの変形で，スワッピングのような計算機タイムシェア

リング・システムのいくつかの特色が付加されている．

図 9-6.1 はシミュレータによってモデル化されたリソースの概略図で，ユーザがそのモデルに対して供給しなければならないパラメータを表にしてある．*MXM* までの個数のトランザクションが同時に主記憶装置 (MS) に駐在してもよいことがわかる．MS と AS の間の 1 つの作業空間の 1 方向のスワップには



リソース・パラメタ

MXM ＝MS内に並行して駐在するトランザクションの最大数
ST　＝ 一方向スワップ時間
H　＝ タイム・スライス
PSV ＝ スワップごとのCPUオーバヘッド時間
V　＝ 0−計算とスワップのオーバラップ無
　　　1−オーバラップ
STS ＝ 0−タイム・スライシング無
　　　1−タイム・スライシング
SC　＝ スケジューラ・コード
RB　＝ MS内に存在しないジョブに対するバイアス値

作業負荷パラメタ

各トランザクション(I)に対して，
A [I] ＝トランザクション要求の時刻
X [I] ＝実行時間

図 9-6.1　シミュレーション・モデル

ST 秒必要であり，1つのスワップを始めるのに *PSV* 秒の CPU オーバヘッド時間を必要とする．オーバラップ・スワッピングを指定し(あるいは指定しない)($V=1$ あるいは 0)，*SC* によって6つのスケジューラのいずれかを指定するが，それがタイム・スライスされるかされないかは変数 *STS* ($=1$ あるいは 0)によって指定される．タイム・スライスは *H* によって指定される．

直接型のタイプ I 作業負荷(第4章参照)では，おのおののトランザクションをその到着時刻と実行時間で示す．**図 9-6.2** は状態図で状態の定義と状態の遷移を示している．**図 9-6.3** はシミュレータの内部で使われているデータ構造のいくつかの概念を与えている．これは，図 9-6.2 の状態図と共に，シミュレータの論理がどのように編成されているかをあからさまにするのに役立ちうる．

次にタイム・スライスを受くべき，あるいは，MS へあるいは MS からスワップされるべきトランザクションの選択をするスケジューリングの決定は，各タイム・スライスの終了あるいはトランザクションの終了時に行なわれる．スケジューリングは2つの部分で考察されるが，これらの間には単純で興味あ

トランザクションの状態
1. 未到着
2. 到着：AS内
3. チャネル内：ASからMSへ (スワップイン)
4. 到着：MS内 (CPUは実行していない)
5. CPU実行中(MS内)
6. チャネル内：MSからASへ (スワップアウト)
7. トランザクション完了：MS内
8. トランザクション完了：AS内

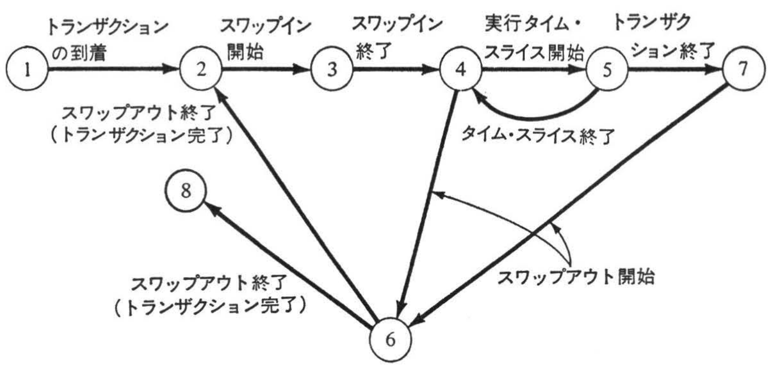


図 9-6.2　一つのトランザクションに対する状態定義と遷移図

トランザクション ──→

		1	2	3	4	等
到着時刻	A					
実行時間	X					
状態ベクトル	S					
実行時間完了	P					
最終スライスの時刻	L					
完了時刻	Q					

（a）トランザクション変数

		Alu. 1	チャネル 2
完了時刻	CV		
サービス中のトランザクション	ID		
利用時間	U		
必用とされる実行時間	XT		

（b）リソース変数

図 9-6.3　いくつかの内部シミュレータ変数

る双対関係がある．第1の部分は，**アドミッション**(admission：入るのを許すこと) 規則と呼ばれ，どのトランザクションが次のタイム・スライスを受ける**べき**かを決める．これは，サービスを受ける資格があるトランザクションの中でスケジューリング変数が最小値を持つものを見つけ出すことによって，なされる[われわれはまたある適切な**分解方式**(tie-breaking rule)を仮定する]．トランザクションが識別されたときには，2つの主な場合がある．すなわち (1) そのトランザクションは MS 内にある，および (2) そのトランザクションは MS 内にない(それゆえ AS 内にある)，である．第1の場合には，そのトランザクションには次のタイム・スライスが与えられる．第2の場合には，もし MS 内に空の作業空間があれば，選択されたトランザクションの“スワップイン”がその空間に対して始まる．スワップインの間，もしオーバラップ・スワッピング・オプションが指定されているなら，スケジューラは MS 内のトランザクションにアドミッション規則を適用し，スワップイン中のタイム・スラ

イスに対して最もよいものを選ぶ.

MS 内に空きの空間がない場合には，スケジューラは，まず，スワップアウトされるべきトランザクションとして，MS 内で完了したトランザクションを探す．そのようなトランザクションが見つかれば，それをスワップアウトし，ふたたび，MS 内のトランザクションにこの期間の間タイム・スライスが与えられる．もし MS 内に完了したトランザクションがなければ，システムは**相補置換え**(complementary replacement)**規則**に照らして，選択されたトランザクションのための余地をつくるために MS から AS へスワップアウトされるべきトランザクションを選ぶ．この規則は次のように働く．すなわち，アドミッション規則をすでに MS 内にあるトランザクションに適用し，**もっとも低い**順位のトランザクションをスワップアウトされる(置換えられる)べきものとして選ぶ．スワップアウト中は，MS 内の別のトランザクションにタイム・スライスが与えられてもよい．ひとたびスワップアウトが完了すれば，事実上 MS 内に空の作業空間が存在し，上述のアドミッション規則が適用可能となる．相補置換え規則は単にただ1つのスケジューリング・アルゴリズムではなく，むしろいくつかの置換え規則のスケージューリングについて述べたものである．もっと明確にいえば，それは与えられたアドミッション規則から置換え規則を得る方法について記述している.

図 9-6.4 はシミュレータの論理の流れ図である．シミュレータはこれまで述べてきたように種々のシステム・パラメータの相互関係について学ぶための有用な教育的な仕掛けである．作業負荷は数値対のストリームとして記述することができる．ここに各対は各トランザクションに対する到着時刻と実行時間を示している．おのおのにつきごく少量(たとえば10個)のトランザクションをもつ2～3のそのようなストリームを合成することによって，ユーザは，異なるスケジューリング・アルゴリズム，MS 内の一時に1つ以上の作業空間，スワップ時間などの効果についての良好な感触をつかみ取ることができる.

われわれはいまや1つのシミュレータの論理についての記述を終ったが，このシミュレータはパラメータの選択によってあるクラスのタイムシェアリング・システムをシミュレートすることができる．さて，このシミュレータをどのように使うことができるであろうか？この重要な質問に答えるためにいかなるシミュレーション・モデルの設計者によっても取扱われねばならない2つの種類の活動を識別することに取りかかろう。

1. シミュレータを駆動する作業負荷についてのデータを得ること.

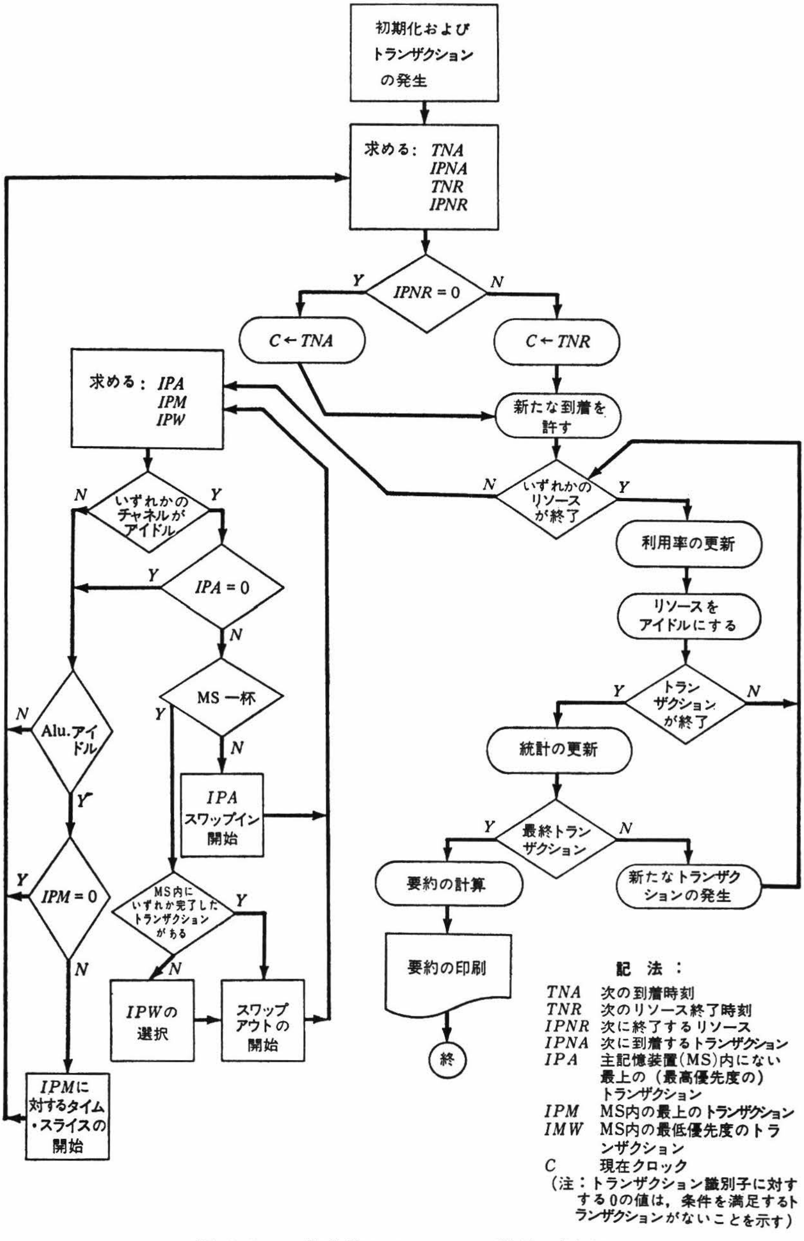


図 9-6.4　基本的シミュレータ論理の概略図

2.　シミュレータが正しく有用な結果を与えるという確信を持つこと．

先に述べたシミュレータは，パラメータの選択によって，APL のリソース管理(9-4 節参照)の論理を近似するように構成することも可能なので，このシステムを以下の例として用いるであろう．

シミュレータを含んだいかなる実用的な評価手段も，システムが実際に遭遇する作業負荷のある現実的な表現を用いて，駆動されなければならない．そのようなデータの理想的な源はソフトウェア・モニタである (8-5 節参照)．われわれのシミュレータは到着時刻と実行時間を必要とする．APL システムは，**ヒストグラム**，すなわち種々の事象の出現頻度分布を供給するソフトウェア・モニタを含んでいる．これらの 1 つに，**思考**時間，すなわちユーザの要求(トランザクション)の間の到着間隔時間の異積頻度分布がある．1 つの測定結果の分布が**図 9-6.5** に示されているが，これには同じ平均値を持つ指数分布 (第 5 章参照) も示してある．この 2 つの関数はかなり接近して一致していることがわかる．

図 9-6.6 は，**実行時間**の測定されたヒストグラムの同様な図を示し，これにも指数分布の図示を一緒にしてあるが，測定結果の分布との一致は悪いことがわかる．これは測定結果の実行時間分布の“長い尾部”によるもので，これか

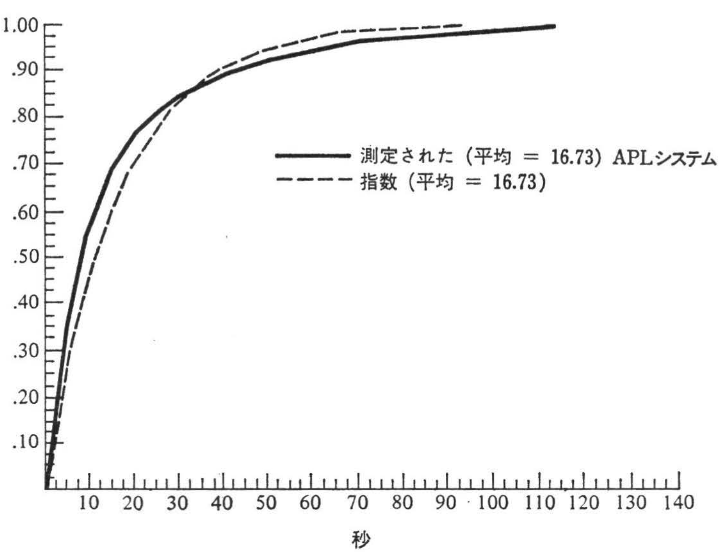


図 9-6.5　思考時間の累積相対頻度分布

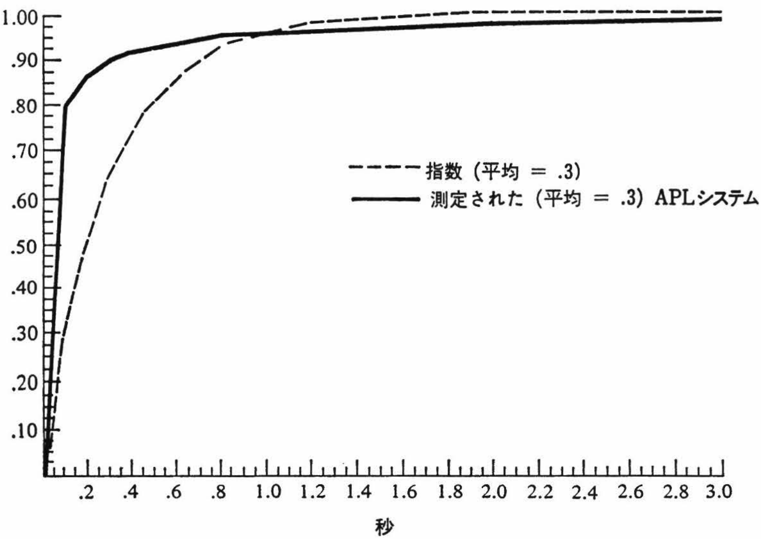


図 9-6.6　実行時間の累積相対頻度分布

ら平均の数倍の実行時間が指数法則によって予測されるよりもずっと多く測定データ中に存在していることがわかる．したがって，たとえば，指数法則は実行の1%以下が1.8秒(平均の6倍)より長いと予測しているが，測定データは実行の約3%がこのカテゴリに入ることを示している．もちろん，分布の尾部はトランザクションの**総数**の小部分を示しているにすぎないが，一方，それはトランザクションによって要求される全実行**時間**のかなりな部分を表わしていることも多い．この理由で，それはリソース使用において重要な意味をもつ．図9-6.6に見られるように，測定結果の分布は同じ平均値を持つ単一指数分布には整合しないが，約4項を持つ超指数分布にはよく適合する可能性があることがわかっている．

図9-6.5と図9-6.6の思考および実行時間分布関数は，シミュレートされた各トランザクションに対する到着時刻と実行時間を供給するために，(2章で述べたように)プログラム乱数発生器によって利用されうる．

さて，シミュレータを駆動するデータを手もとに持ったので，先に示した第2の点，すなわちシミュレータの論理に確信を持つこと，に取りかかろう．これを行なう1つの方法は，それが数学的解析によって得られるものと同一の結果を与えることを確かめることである．この種のテストは，もちろん，シミュ

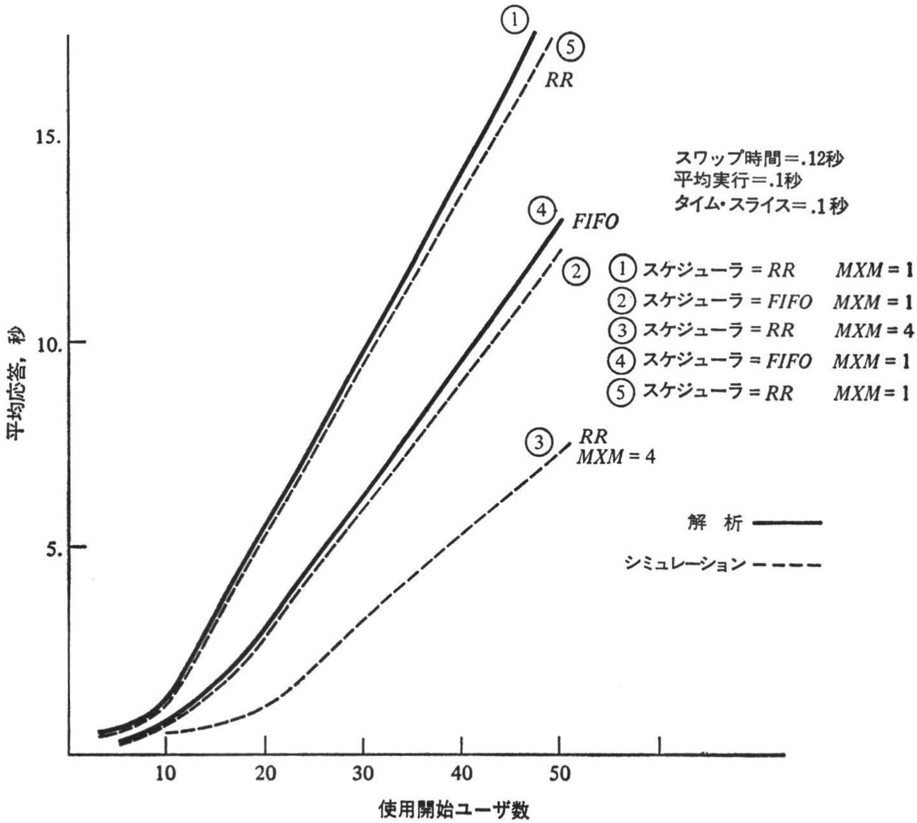


図 9-6.7 シミュレーション結果と解析結果の比較

レータの論理にとって必要ではあるが十分では**ない**テストである．というのは，それには，普通，モデルの縮退が伴うこと，すなわち解析的な解が存在するように非常に簡単化した場合のモデルを用いることになるからである(5-8節)．図 9-6.7 は，このような方法でシミュレータを使用した結果を示している．2 つのスケジューラ，RR および FIFO が，指数到着分布および指数実行時間分布で，使用された．解析的およびシミュレーション結果は非常に近いことがわかる．

シミュレータに確信を持つためのもう 1 つのそしてより重要な方法は，その結果を**測定**結果と比較することである．ここでなされるべき最初の選択の 1 つは，比較されるべき変数の値の組である．たとえば，性能を決定するシステ

ム・パラメータの多くに対してむしろ**鈍感な**(あるいは，統計家がいうように，**頑健な**)変数を選択することができる．その場合は，測定とシミュレーションとの間に良い一致を得るのは比較的容易である．逆に，比較のために，多くのシステム・パラメータに非常に鋭敏な変数を選択することができる．この場合には，もはや優良な一致は困難であろう．というのは，モデルとシステムの間の小さな差が選択された変数値では大きな差を生むであろうからである．以下において，われわれは，(1)（タイムシェアリング・システムの目的にとって）最も意義のある性能インジケータで，(2) APL システムからの測定値として利用できる，変数を選んだ．これらの変数はシステム・パラメータに非常に鋭敏であるので，測定とシミュレーション結果の間にはかなりな差が予想されまた観察もされる．値の間の実際の一致よりももっと重要なものは，おそらく関数の形が近いことと傾向をよく追っていることであろう．これらすべてがすぐ見られるようにかなり良好である．測定は APL のみを 55 人のユーザの使用開始で走らせた IBM システム/360 モデル 50 上で行なわれた．

図 9-6.8 は測定とシミュレーションによる反応時間分布である．**反応時間**はトランザクションの到着と最初のタイム・スライスの間の時間として定義される．図 9-6.9 は，実行/経過時間，すなわち“伸張係数”の逆数の累積相対分布を示す．したがって，たとえば，実曲線上の点(10%，.7)は“(測定された)

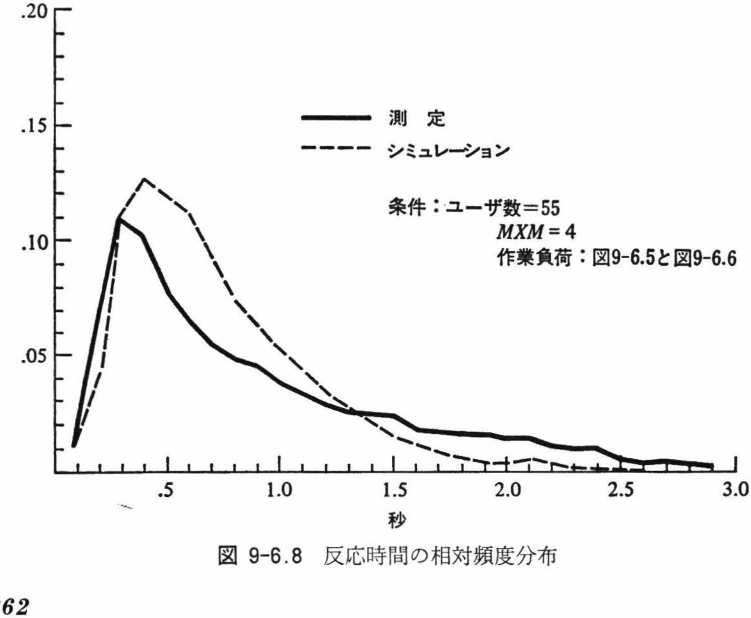


図 9-6.8　反応時間の相対頻度分布

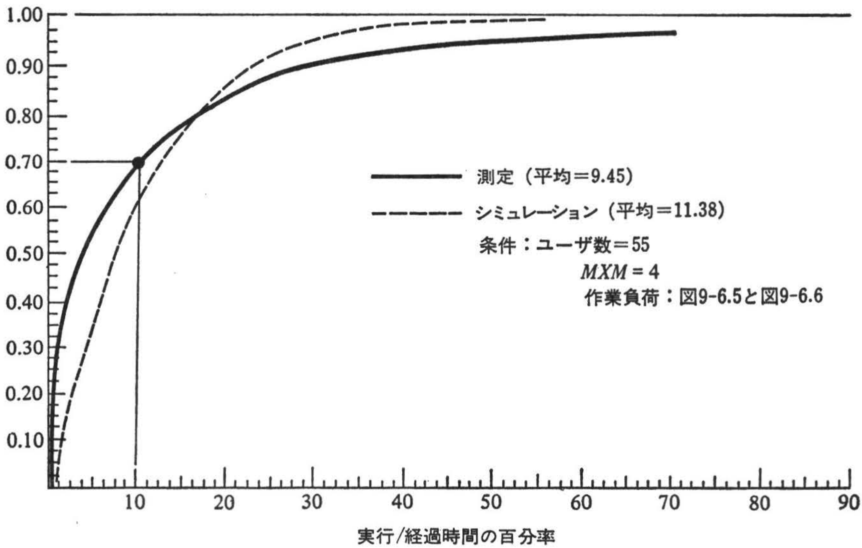


図 9-6.9　実行/経過時間百分率の累積相対頻度分布

トランザクションの70％は.1以下の実行/経過時間比を伴った" ことを意味する．いいかえれば，トランザクションの70％は10倍以上の伸張係数を伴ったのである．

図9-6.8および図9-6.9は両方とも，すべてのトランザクションを一緒にした性能測定を与えている．前の議論から，タイムシェアリング・システムの成功は，軽微トランザクション(およびその他の短いもの)に迅速にサービスする能力に最もきわどく依存していることを思い出そう． APLはこれをどのぐらい適切に行なっているであろうか．図9-6.6から，トランザクションの約80％は1タイム・スライス(このシステムでは.1秒)以下の実行時間を必要とするカテゴリに入るということに注意するとある考えが浮かんでくる． 図9-6.8は，ほとんどの要求は1秒以内に**最初の**タイム・スライス(軽微トランザクションが必要とするのはたった1つのタイム・スライスである)を受けとることを示している．これらの事実はAPLが軽微要求に対して非常に応答がよいということを有力に示している． この特性は(長い非軽微トランザクションに対する応答と違って)使用開始ユーザの数のみに伴って非常にゆっくりと下がってゆく．

図9-6.10は，主記憶装置内により多くの作業空間を保持すること(但しスワ

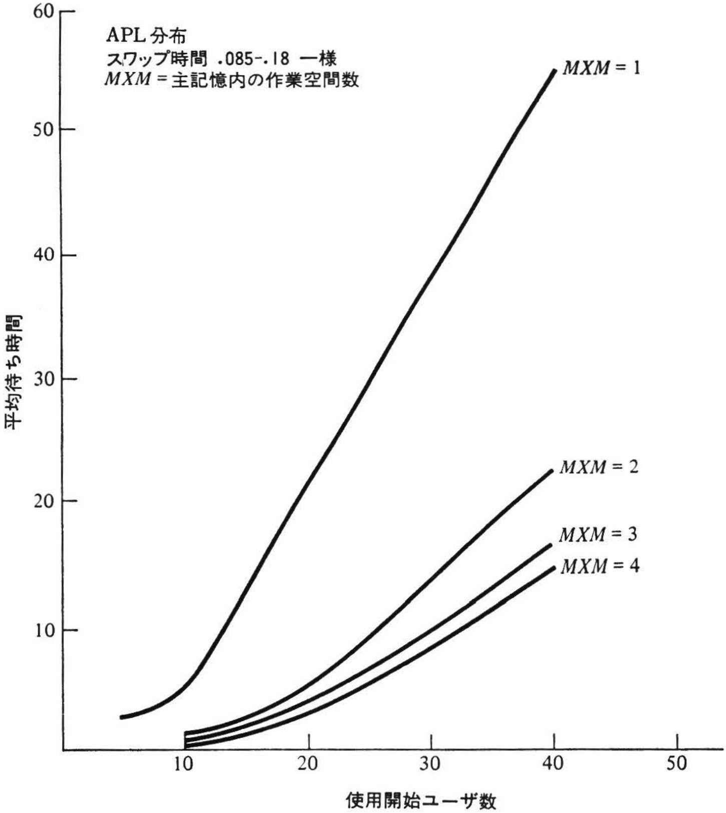


図 9-6.10　主記憶内の種々の作業空間数での APL 性能のシミュレーション

ッピング・チャネルは2つのみ)の効果を測るためのシミュレータによる研究の結果を示している．同一使用開始ユーザ数に対して，駐在する作業空間を1から2にすると性能は大きく改善されるが，それを超える(たとえば *MXM* を3から4にする)改善は限界にきている．

図 9-6.11 は，

1. 使用開始ユーザ数
2. CPU 速度係数
3. 駐在する作業空間の数

についていくつか値を選びその変動の，平均待ち時間，待ち行列，新規待ち行

使用開始ユーザ数	30	30	50	50	30	30	50	50
CPU 速度係数	4	4	4	4	1	1	1	1
MXM (MS 内の最大数)	2	3	2	3	2	3	2	3
平均思考時間	16.75	16.75	16.75	16.75	16.75	16.75	16.75	16.75
平均実行時間	0.08	0.08	0.07	0.08	0.28	0.29	0.21	0.25
平均待ち時間	0.44	0.34	1.30	0.73	1.37	0.80	5.69	2.44
平均待ち行列長	0.85	0.66	0.82	2.2	2.86	1.82	15.35	7.48
平均新規待ち行列長	0.5	0.5	2.0	1.9	0.7	0.6	1.3	1.7
フィギュアオブメリット	0.41	0.43	0.18	0.21	0.37	0.42	0.17	0.20
スループット	2.99	3.28	2.96	3.40	1.95	2.28	2.05	2.58
% CPU 利用率	17	17	27	28	52	55	58	79
%チャネル利用率	50	46	84	77	63	54	88	89
%オーバラップ	9	9	17	20	33	34	45	69

この表におけるすべてのランに対する条件

トランザクション数/ラン = 2000

作業負荷 = APL 分布(図 9-6.5 および図 9-6.6 を参照)

タイム・スライス = 0.1 秒

スワップ・タイム = 0.025 から 0.25 秒

スワップあたりの CPU オーバヘッド = 0.01 秒

チャネル・インタフェース = 0.15

図 9-6.11 代表的なシミュレーション結果

列(新規の到着), フィギュア-オブメリット(短ジョブ), スループット, リソース利用率, およびオーバラップに対する効果を示す一覧表である. これは, 主要なシステム設計パラメータについて判断する際に有用な摘要表示の典型である.

9-7 IBM システム/360/370 のタイムシェアリング・オプション (TSO) [IB 1, IB 2, IB 3, LE, MA, SC 2 参照]

1971 年に最初に発表されたこのシステムの主な考え方は, タイムシェアリング・モードに IBM の汎用 Operating System (OS/360) の広範な機能を提供することである. また TSO はそのインタラクティブ・ユーザに, 特にタイムシェアリング環境のために設計されていて, バッチ・システムには提供されない, 一定の言語機能にアクセスすることを許している. しかしながら, ユ

ーザはOSバッチと完全に互換性のあるプログラムを開発することにしてもよい．バッチのランを端末から始めてもよいし，またシステムはタイムシェアリング(フォアグラウンド)ジョブにサービスしながらバッチ作業負荷を走らすこともできる．

さて，TSOを次の2つの関係において簡単に記述しよう．その第1は各個々のユーザに利用可能な機能を考慮する．その第2はリソース管理・共有スケジューリングの面に及ぶ．

単一ユーザ機能

各ユーザは最初**コマンド言語**を通して対面する．この言語はバッチ・システムにおけるジョブ制御言語と同一の機能を遂行するが，タイムシェアリング環境のために拡張されそして“人間化”されている．コマンド言語は次のものを含んでいる．

1. **データ・セット**(data-set)と呼ばれる格納情報の入力，格納，表示，および修正に便利なEDIT(編集)コマンド．編集機能は**文脈探索**(context-search)と行番号参照の両方を含む．
2. プログラム名に対して格納記憶対象や装置を後になって結合できるように，データ・セット名とプログラム内に使われている名前(ファイル名)とを関連づけるコマンド．
3. プログラムの実行．

コマンド・プロシジァ(すなわちコマンド言語におけるサブルーチン)としてコマンド順序列を格納する能力は意義のある特徴である．これらは，OS JCL-カタログ・プロシジァ(8章参照)と対をなすその片方のものである．おのおののそのような順序列には1つの名前が与えられ，また通常のサブルーチン・リンキングやマクロコールとほとんど同じようにしてパラメータが供給されてもよい．実際，TSOコマンド言語の構文はOSアセンブラ・マクロ言語によく似ている．コマンド・プロシジァは位置パラメータとキーワード(自己識別)パラメータの両方を処理してもよい．後者には“標準”値(すなわち，欠値)が割付けられていてもよいであろう．例としては，(たとえば) PLICLGO (この名前はプロシジァ自身には含まれていない) と呼ばれる格納コマンド・プロシジァの**ヘッダー**を考えよう．

PROC 1, & X, SYS1(*), SYS2(*)

このプロシジァを書いた人は，“1”によって，1つの位置パラメータがあることを指定している．その名前は X である(第1番目の名前はその数につづく)．SYS1 と SYS2 はキーワード・パラメータ名で，それらの欠値は * である．* は装置として端末を表わすシンボルである．したがって，ユーザは(PLICLGO と名づけられた)このプロシジァを

PLICLGO STAT SYS1 (DATA)

あるいは

PLICLGO STAT

のようなステートメントによって呼んでもよい．第1の場合には，このプロシジァが呼ばれると，プロシジァ内に現われるすべての X は名前 STAT に置換えられ，またプロシジァ内に現われるすべての SYS1 には名前 DATA が使われるであろう．このコールでは SYS2 については何も言及されていないので，プロシジァ内でそれが現われるたびに，その欠値 * (端末を意味する)が使われるであろう．要約すると，上記の第1形は PLICLGO と名付けられたコマンド・プロシジァを呼ぶが，これは上記の PROC ヘッダーを持っており，結果的に STAT と名付けられたデータ・セット内に格納されたプログラムの(たとえば)コンパイルおよび実行を行なうことができる．このプログラムは DATA と名付けられたデータ・セットからの入力データを予期しており，そして端末にその出力を与える．上記のコールの第2の形も同様であるが，SYS1 もまた省かれているので，端末に対しては出力データ装置のみならず入力データ装置も共に指定を欠いているわけである．

格納コマンド・プロシジァの考えは非常に重要である．それは一般によく使われている順序列を繰返すという退屈さをなくするのみならず，それによって，システムの内部操作についてほとんど興味も知識もない応用向きユーザのために言語サブシステムを作って供給することもできる．そのようなサブシステムは IBM あるいはもっと一般には導入設置システムのプログラマによってプログラムされてもよい．通常のユーザでさえ，望むならば，自分自身のコマンド・プロシジァを構成してもよい．

この著書を書いている時点(1975)では，TSO のコマンド言語での経験からその大きな潜在能力が示されているが，その構成上のある見落しによって現在の所は，それがややだいなしになっているのである．TSO は，随意課金で，インタラクティブ使用を意図した形の3つの言語，ITF/BASIC，ITF/PL/I，

および**コードアンドゴー**(code-and-go) FORTRAN を提供する．すべて，入力時での行ごとの構文チェックおよびその他のデバッグ機能を特色とする．

FORTRAN, COBOL, PL/I, ALGOL, およびアセンブラの標準 OS/360 コンパイラもまた利用可能である．OS 順序アクセス法(BSAM および QSAM)は端末向けのものといえる．いくつかのプロセッサがタイムシェアリング環境向きにいくらか変更した形で提供されている．TEST コマンドによって，端末からラン中のプログラムの START/STOP およびトレースができる．不幸にして，この手段はソース言語デバッギングを提供しない．出力は機械言語であるので，ユーザは結局シンボル表を用いて TEST 出力をシンボリック出力に変換しなければならない．

TSO† におけるリソース管理

TSO におけるリソース管理を理解するための手始めとして，**図 9-7.1** を考察しよう．この図は論理階層中に現われる主要な対象を示している．中央リソース・スケジューラは**ドライバ**と呼ばれる．それは，**タイムシェアリング・インタフェース・プログラム**を通じてすべての他の主要な制御機能と通信する．ドライバはタスクとリソースの状態の両方についての報告を受取り，そして，また導入時に指定される一定のパラメータを用いて，どのジョブを記憶装置に対してスワップイン・スワップアウトすべきか，またそのおのおのが CPU 時間をどのくらい受けるべきか(これについては後にもう少し詳しく述べる)を決定する．ドライバはまた作業負荷のバッチ(バックグラウンド)成分に関する主要な制御をつかさどる．

システム制御の流れは，**図 9-7.1** を用いて大ざっぱにたどることができる．システム・オペレータが MVT をスタートした直後で，TSO をスタートする以前のシステムを考えよう．オペレータはタイム・シェアリング制御タスク(TSC)と呼ばれるシステム・プログラムを開始するコマンドを与えて TSO をスタートさせる．このタスクは(パラメータを受けとるために)オペレータとさらに通信することができ，また**各**フォアグラウンド・リジョンのためのリジョン制御タスク(RCT)を開始することができる．各 RCT は，次に，端末から入れられた LOGON コマンドに応答して LOGON スケジューラの写しを呼ぶことができる．LOGON スケジューラはユーザの LOGON メッセージ内の情報を受理し，またユーザ属性データ・セット (UADS) と呼ばれるデータ・セ

---

† この話題に関するより詳しいことについては，文献 IB 3，pp. 37-52 を見よ．

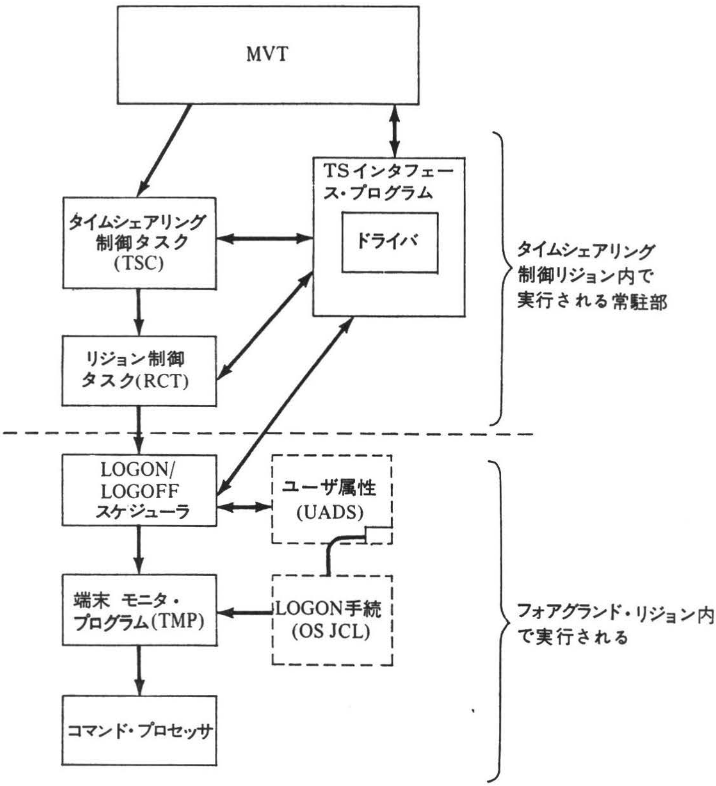


図 9-7.1 TSO 内の制御の論理的階層

ットにアクセスする．このデータ・セットはおのおのの認定ユーザに関する了め格納された情報(課金番号，LOGON 合言葉など)を含んでいる．UADS には LOGON プロシジァ(手続)と呼ばれる OS JCL (ジョブ制御言語) カタログ・プロシジァの名前が含まれている．LOGON プロシジァは単一の OS ジョブ・ステップを規定するが，この理由によって，どの単一の端末においても LOGON から LOGOFF までの各 TSO セッションは単一ステップの OS ジョブとみなされる．8章から，1つのジョブ・ステップは JCL 内で，1つの EXEC ステートメントとそれに続くいくつかの DD ステートメントによって，指定されることを思いだそう．EXEC ステートメントは実行されるべき一つのプログラムを指定する．LOGON プロシジァ(図 9-7.2)の場合には，このようにして呼ばれたプログラムは，セッション中ユーザが入力するすべての

```
//NFSPROC    EXEC   PGM = IKJEFT01, ROLL = (NO, NO)
//SYSUDUMP   DD     SYSOUT = R
//SYSHELP    DD     DSN = SYS1. HELP, DISP = SHR
//SYSEDIT    DD     DSN = &EDIT, UNIT = SYSDA, SPACE
                                        = (1688, (50, 20))
//SYSPROC    DD     DSN = SYS1. CMDPROC, DISP = SHR
//DD1        DD     DYNAM
//DD2        DD     DYNAM
//DD3        DD     DYNAM
//DD4        DD     DYNAM
//DD5        DD     DYNAM
//DD6        DD     DYNAM
//DD7        DD     DYNAM
//DD8        DD     DYNAM
//DD9        DD     DYNAM
//DD10       DD     DYNAM
```

図 9-7.2　典型的な LOGON プロシジャ

端末コマンドを解釈するためのものである．この解釈プログラムは特定の目的のためにユーザ自身によって書かれてもよいが，ここでは，そのプログラムは IBM によって供給されたもので，端末モニタ・プログラム(TMP)と呼ばれるものであると仮定しよう(ということはよくあることだが)．TMP は“ローカル・スーパバイザ”と考えてもよい．というのは，それはユーザ・コマンドを解釈し(それは構文解析用のコマンド・インタプリタを呼んでもよい)，そのコマンドを満足するのに必要などのプログラムでも，それをコールする過程をスタートさせるからである．EXEC ステートメントに加えて，LOGON カタログ・プロシジァはユーザ・セッションの間に必要とされることになる(ディスク空間の予約およびデータ・セットへの名前の結合のために)いくつかの DD ステートメントを指定する．各ユーザは自分のカタログ内にいくつかの LOGON プロシジァを持ってもよい．それらの名前はユーザの UADS に格納され，ユーザは自分の望みのものを LOGON メッセージの中で指定する(あるいは指定なしで供給される)．LOGON と関連したこれらすべての事柄は各ユーザに“各自の”論理マシンを指定させ，スタートさせるための1つの方法と考えてもよい．

図 9-7.3 は論理対象のおのおのの要約である．**図 9-7.4** は，1フォアグラウンドおよび1バックグラウンド・リジョンを持つ典型的な TSO システムにお

対　象	略　称	型	数	活動駐在域	主　要　機　能
ドライバ	—	システム・プログラム	1/システム	T. S. 制御リジョン	TSO およびバッチの主リソース・アロケータ
タイムシェアリング・コントロール・タスク	TSC	システム・プログラム	1/システム	T. S. 制御リジョン	1. RCT の開始 2. TSO のシステム・オペレータとの通信 3. スワッピングの制御
領域制御タスク	RTC	システム・プログラム	1/フォアグラウンド領域	T. S. 制御リジョン	1. 静止化制御 2. LOGON/LOGOFF プロシジァの開始
端末モニタ・プログラム	TMP	システムあるいはユーザ・プログラム	1/使用開始ユーザ	フォアグラウンド・リジョン	端末から直接来るコマンドを解釈し，コマンド・プロセサを呼ぶ
ユーザ属性データ・セット	UADS	データ・セットの分割	1/認定ユーザ	ディスク	1. LOGON プロシジァの名前 2. 課金番号，LOGON 合言葉 3. ユーザに関する情報
LOGON プロシジァ	—	OS JCL カタログ・プロシジァ	1あるいは1以上/ユーザ	ディスク	TMP を開始し，端末セッションにおいて必要なデータ・セット(プロシジァを含む)を指定する
コマンド・プロシジァ	—	データ・セットの分割	いくつでも	ディスク	格納されたコマンドのサブルーチンで 1. 実行すべきシステムおよびユーザ・プログラムを呼ぶ 2. 空間を確保する 3. データ・セット(物理空間の名前)とファイル(プログラム名)を結合する

図 9-7.3 TSO における重要なシステム対象

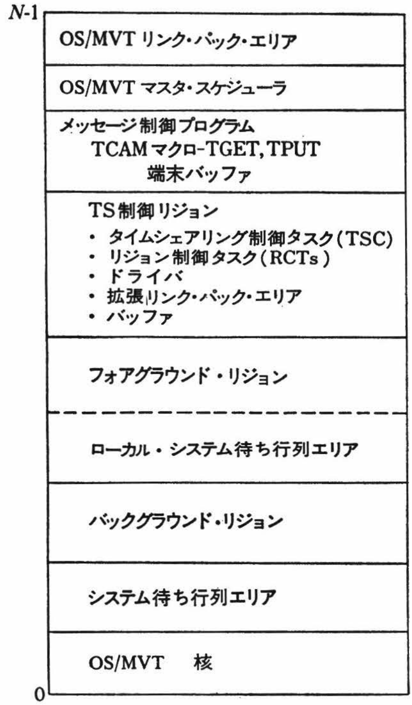


図 9-7.4　典型的な TSO システムに対する主記憶装置の割付

ける主記憶装置の割付を示す．エリアの大部分は OS MVT によって必要とされるものであることに注意しよう．その新しいものには，常駐 TSO システム・プログラムの大部分を保持するタイムシェアリング制御リジョンおよび，当然，フォアグラウンド・リジョンが含まれている．後者は2つの部分に分割されていることに注意しよう．すなわちその1つは上から下へ割当られ，他の1つは下から上へ割当られる．これらの第2のものは**ローカル・システム待ち行列エリア**と呼ばれ，そのリジョン内の現在の活動と関係を持つ制御ブロック情報を格納するために確保される．これは，そのリジョン内のその他の有効な情報と一緒にスワップアウトされる(そして後にスワップ・バック・インされる)であろう。(そのリジョンのうち有効な情報を含んでいる部分のみがスワップされる.)

1つの主要な性能の論点が図 9-7.4 を用いて議論できる．タイムシェアリング制御リジョン内の**拡張リンク・パック・エリア**は TSO ユーザによって必要

とされる共用のシステム・プログラムのコピーを保持する．普通の（非仮想記憶方式の）OS の場合と同様に，おのおのの導入設置にあたってはこの目的のために使用される空間の大きさおよびそこに常駐させるべきルーチンとデータの種類を指定しなければならない．ここに常駐していないルーチンはいずれも，使用のたびにディスク（ドラム）から取ってこなければならないので，高い活動度のモジュールの選択は良い性能を得るための本質である．メーカは導入システムが運転開始にいたるのを助けるためにいくつかの示唆をする．しかしながら良い性能を得るには，常駐させるべきモジュールの最もよい集合を見分けるためにソフトウェア・モニタを利用する必要がある．このことは，おそらく，高度にモジュール化され複雑なデータとプログラムの構造を特色とするOS 様のシステムにおける，主要な性能の論点である．もし常駐対象がよく考慮して選ばれていないと，OS のデータ管理のための（TSO の部分ではなおさら多い）I/O 活動が，スワッピングよりも装置競合に大きな役割を果たすことになるかもしれない．

さて，ここでもう1つの重要な性能の論点に取りかかろう．すなわち，主記憶装置に入れる端末ジョブのスケジューリング（**主タイム・スライシング**あるいはサービスと呼ばれている）とすでに主記憶装置内にあるジョブ（バックグラウンド・ジョブを含む）の間の CPU のスケジューリングについて論じよう．TSO の制御プログラム（ドライバおよび TSC）は，フォアグラウンド・リジョンとスワップ装置の間の情報の移動をスケジュールする．あるリジョンの実際に使用される部分のみが――それはスワップ・ロードと呼ばれるが――実際にスワップされる．そして**スワップ・アウトはそのリジョンに関係する進行中のすべての非端末 I/O が完了した――静止化**と呼ばれる過程――時にのみ開始される．（APL において行なわれているように）各端末に対するバッファ空間は主記憶装置内に常駐的に取られているので，ユーザのプログラムがスワップアウトされている時でも端末メッセージは進行できる．したがって，I/O 端末を静止化する必要はない．

きわめて重要な CPU および主記憶リジョンのリソースのスケジューリングは，2つのスケジューリング時間間隔に集中している．その第1はジョブの**主タイム・スライス**と呼ばれるもので，ジョブがスワップインから次のスワップアウトの間主記憶装置に滞在できる時間として定義される．その第2はジョブの**副タイム・スライス**と呼ばれるもので，現在駐在しているジョブに対してCPU が割当てられる最大連続時間である．両方の型のタイム・スライスともいくつかのパラメータに依存するが，それらのあるものは導入設置時に（シス

テム・プログラマによって）供給され，またそれらの他のものはそのシステム自身によって測定される．特定のパラメータ値の集まりは**論理待ち行列**と呼ばれる．共通の論理待ち行列を持つユーザ・ジョブの集まりは，サービスされるべきジョブの1つの待ち行列を構成する．TSOにおいては，“あるジョブにサービスする”ということは，単にそれを主記憶装置にスワップインするということを指している．したがって，1つの論理待ち行列は，その待ち行列に属するあらゆるジョブの主タイム・スライスを決めるスケジューリング・パラメータの1つの集まりである．1つの主記憶装置のリジョン上にいくつもの待ち行列があってもよいことに注意しよう．いずれの待ち行列においてもそこでサービスを待っているジョブは，ラウンドロビンの順序でサービスされる．しかしながら，TSOはいくつかの待ち行列を持ち，ジョブを異なるサービス等級に割当てるようにすることもできる．論理待ち行列を決めるスケジューリング・パラメータについて，またあるジョブが1つの待ち行列にどのように割当てられるか，さらにそれがどのように待ち行列の間を動かされるかについていくらか詳しく論じよう．その前に，現在主記憶装置内にあり（そして走る用意ができている）ジョブに対する副スライス（すなわちCPUタイム・スライス）のスケジューリングも，後で述べるように，やはりあるスケジューリング・パラメータに依存しているということにれ触れておかねばならない．

上に示したように，おのおのの主記憶装置のリジョンはジョブのいくつかの待ち行列を持ってもよい．与えられたいずれの待ち行列も，手作業によってのみ変更できる次のパラメータに対して導入設置時に割当てられた，固定値を参照することによって管理される．

$A_{R,Q}$ = その待ち行列内のジョブに対する平均サービス時間
$M_{R,Q}$ = 最小主タイム・スライス
$SL_{R,Q}$ = スワップ・ロード限界
$IL_{R,Q}$ = インタラクション時間限界
$SC_{R,Q}$ = 次の待ち行列に進むまでのサービス・サイクルの数

付加パラメータ（$N_{R,Q}$）は，刻々のシステムによって決まってくる．それはその待ち行列内のレディ（用意のできた）ジョブの総数を指す．$A_{R,Q}$はその待ち行列内の与えられたあるジョブの2つの連続したスワップインの間の時間間隔である．$M_{R,Q}$の目的は，その待ち行列上のジョブのおのおのが，ある生産的な仕事を行なうのに必要な時間長主記憶にとどまっていられるように保証することである．$SL_{R,Q}$は，この待ち行列上のジョブが必要とする最大主記憶量

を定めるものである．$IL_{R,Q}$ は，ユーザの最後のサービス要求の後主記憶内に留りうる最大合計時間である．これら4つの値の特定の組は，いずれも，ある1つの等級のジョブのスケジューリング特性の性質を記述している．ある待ち行列に対する上述のパラメータから，スケジューラは，次のように，その待ち行列上の各ジョブに対する主タイム・スライスを計算する．

$$T_{R,Q} = \mathrm{Max}\left(M_{R,Q},\ \frac{A_{R,Q}}{N_{R,Q}}\right)$$

それで，各ジョブは時間 $T_{R,Q}$ の間，主記憶装置上に保持される． その待ち行列内のすべてのジョブがこのようにしてラウンドロビンの順序でサービスされる． 現在の待ち行列上で $SC_{R,Q}$ サイクルたつと， 次の待ち行列がサービスされる．

制限パラメータ $SL_{R,Q}$ および $IL_{R,Q}$ が，もしあるジョブによって超過されると，システムは，そのジョブのパラメタを含むことができるような新しい待ち行列をそのジョブのために探すことになる．新しい待ち行列は同一リジョン上に存在しなければならない． というのはシステム/360 では動的再配置ができないので一度プログラムの実行が開始されるとそれを新しいリジョンに移すことができないからである．しかしながら，そのような新しい待ち行列は，一般に，その他のパラメータに対しても異なる値を指定することになろう．たとえば，次のような2つの待ち行列を考えよう．

	$Q1$	$Q2$
A	1	10
M	.2	1
SL	100 k	100 k
IL	.1	5

ここに，小さな IL および A の値を持つ $Q1$ は，明らかに高度にインタラクティブな要求にサービスするための迅速なスワッピングを意図したものであり，一方， $Q2$ はすでにかなり長いことが判ったジョブに対して意図したものである． 大きな A の値は $Q2$ ジョブを主記憶装置内により長くとどめようとする傾向があるので，それらのジョブがスワップイン間に生産的な CPU の仕事をする機会がより多くなることに注意しよう．

ある与えられた時点で主記憶装置の TSO リジョンを占めていて実行準備のできている(すなわち，I/O 待ちではない)ジョブの1つが，**副タイム・スライ**

スと呼ばれるCPU間隔に対して選ばれる(**ディスパッチ**される). TSOでは, そのシステムを走らせているシステム係員は全CPU時間のある保証された割合をバッチ作業負荷のために指定することができる. 残りのCPU時間は**フォアグラウンド・タイム**と呼ばれ, これから述べようとするのは, この時間の副タイム・スライスへの再分割である.

フォアグラウンドCPU時間に対して利用できる時間は, (システム・プログラマによって選ばれる) 3つの機構のどれか1つによって副タイム・スライスに割当てられよう.

1. **単純ディスパッチング**: 副タイム・スライスは, フォアグラウンド・タイムに等しく設定される. この時間は, もっとも優先度の高いジョブ(すなわち最も以前にスワップされたジョブ)に対して与えられる. **単純ディスパッチ**は, そのシステムでただ一つのフォアグラウンド・リジョンしか使われていないときには, 常に利用される.
2. **一様ディスパッチング**: 副タイム・スライスは, フォアグラウンド・タイムを主記憶装置内のレディ状態のフォアグラウンド・ジョブの総数で割ったものである.
3. **重み付きディスパッチング**: システムは, 各ジョブに対して, I/O待ちに関して (スワップインを要求しない端末I/Oを勘定に入れないで) その過去の振舞に基づいて, 見積りの待ち時間割合を計算する. 見積りのI/O待ち時間が長ければ長いほど, 計算される副スライスは長くなる. これは, I/Oに束縛された(I/O-bound)ジョブがI/O待ちのために, CPUに束縛された (CPU-bound) ジョブに対比して失うであろうタイム・スライスの部分を補償している.

9-8 G.E. インフォメーション・サービス・ネットワーク[GE参照]

今まで, われわれは多くのユーザが電話回線を通じて結合される単一CPUシステムのタイムシェアリングについて議論をしてきた. そのようなシステムは通常ある単一の組織の必要に対してサービスする一般利用のものであるが, そこでは同じ設備がしばしばバッチ作業負荷とで共用されている. ある会社が加入制を基にして端末を手にするどの人に対してもタイムシェアリング・サービスを提供するという場合は, やや異なった型のサービス販売方式である.

General Electric Companyの**インフォメーション・サービス・ネットワーク**は, 多くのリモート・ユーザにサービスするために設計された設備とソフト

ウェア構成から成っている加入制サービスである．これらユーザの部分集合は同一組織に属していてもよいしいなくてもよいし，また(ユーザ・ファイルのような)共通な情報を共有していてもいなくてもよい．その上，(ユーザのみならず) システムのリソースは地理的に広範囲に分散していてもよいし，また加入者に対して最小限の中断をするだけで変ってもよい．そのようなシステムは，時に，**インフォメーション・ユーティリティ**と呼ばれる．というのは多くの点で，それが電力や電話通信などを供給しているような他のユーティリティと似ているからである．1人の加入者とみなされる各ユーザは，自分が使用した設備(中央設備の空間・時間の自分の割り当て分を含んで)に対してだけ支払いをすればよく，システム・リソースの大きな投資，開発および管理の重荷と責任にさらされないようになっている．これらを供給する組織は，満足のゆくユーザ・サービスを提供することによってその収入をかせいでいる．

図 9-8.1 は，通信形態のいくつかの原理を示すためにそのネットワークの断片を示している．ユーザ端からはじめると，適当な電話番号をダイヤルすることによって端末 (図に示されていない) はその地にある小型計算機 (G. E. 235) につながる．いったんつながると，(ハードウェアの観点からは)すべての通信は，(シカゴ，イリノイというような) その地の多くの端末にサービスできる G. E. 235 と直接行なわれる．地方に置かれたこの計算機は，メッセージの**集中装置**としての役を果たす．これは，多くの地方回線から到来する(そして出立する)メッセージを累積(バッファリング)し，最終的には(たとえば，オハイオのクリーブランドにある)中央計算機所在地に到達している一本の長距離**賃借**電話回線を通してそれらのメッセージを伝送する．中央計算機所在地では，もう1つの計算機(4020)がそのメッセージをもとの状態に分解し，エラー・チェックを行ない(そして，もしエラーが発見されたら再伝送を要求し)，そのメッセージを G. E. 635 中央計算機システムに転送する．以上は，端末から計算機へのいわば本国行きのメッセージの立場から述べてきたが，逆のいわば外国行きの方向についても同様な過程が行なわれる．この章のまえがきでも述べたように，回線集中装置を用い一本の賃借回線を効率的に共有することによって電話料金を十分切詰めることができる．各ユーザは地方電話料金のみを直接支払うのである(賃借回線の分担金は，加入料金の中に含まれる)．

図 9-8.1 から明らかなネットワークの他のいくつかの特徴は注目に値しよう．同一市内の加入者は多くの集中装置計算機――そのおのおのは特定の中央設備に直接つながっている――のいずれか1つに到達できればよい．このことは，図ではシカゴの2台の集中装置の場合に見られる．おのおのの中央集中装

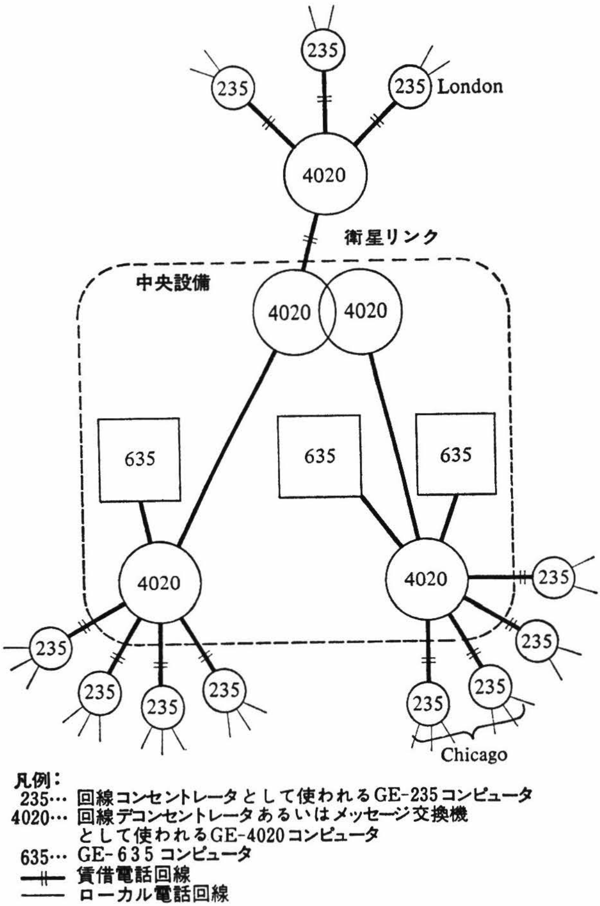


図 9-8.1　G. E. 情報サービス・ネットワークの断片の概略図

置計算機(4020)は，各計算機処理設備が他のどれとも通信できるような形で，他と接続されており，各ユーザの端末は原則的にいくつかの中央計算機のいずれにも到達できる．

ネットワークの潜在的に大きな利点の1つは，リソースをプールする能力であり，そこではリソースは実際の要求が現われるのに応じて違ったやり方で分配されうる．考えの上ではこのことを行なう多くの方法がありうる．ここではG. E. ネットワークのためのそれらのいくつかに触れよう．この本を書いてい

る現時点では(1972年中頃)，各加入者の課金番号は，特定のG. E. 635システムの所在地にあるファイル記憶装置と関係している．この関連はシステムによって自動的に行なわれるので，ある与えられた加入者番号に対するサービスは，システム間で自動的には移動され**ない**．しかしながら，この番号の割当てを行なうに当たって，加入者番号を比較的軽い負荷にある635に割当てることによって，システムの負荷平衡をはかろうとする試みがなされる．もう1つの有効な負荷平衡の要因はいろいろな地理的位置の間の時差(ニューヨークはロサンゼルスより3時間進んでいるというように)である．営業日にしたがって活動の時間が少しずつずれている．

これまでわれわれはG. E. ネットワークの通信の面についてのみ議論してきた．ここでしばらく，前に議論したAPLあるいはTSOシステムには見られなかった特徴に重点を置いて，個々のユーザにとって利用できる機能について述べよう．

現在，このシステムは3つの言語，BASIC, FORTRANおよびREMAPTを提供している(以前のものは，ALGOLも提供していた)．BASICはFORTRAN様の言語である．これはDartmouth大学で最初設計され，その後ファイル管理機能および(異なる型の間での)文字列の操作命令を含むようにかなり拡張された．BASICおよびFORTRANともコンパイラ・トランスレータを用いる．REMAPTは，部品を記述するのに便利のよい言語である．その記述は数値制御工作機械のためのプログラムに翻訳される．中央のプログラム・ライブラリはすべてのユーザに利用可能であり，それには，統計プログラム，科学計算プログラム，経営管理ゲーム，その他が含まれている．便利な**テキスト・エディタ**機能があり，ソース・プログラムおよびデータを容易に作り出したり，表示したり，修正したりすることができる．かなり広いファイルをディスク上に保持できる(ユーザは1月あたりのキロバイトの単位で使用料を払う)．プログラムとデータの両方を保持する**ファイル・システム**は，安全性およびプライバシイの点について慎重な注意が払ってあるが，これは注目に値することである．たとえば，あるファイルがユーザによって抹消されたとき，この空間を割当てられた新しいユーザがそのファイルをうっかり読まないようにするため，その内容の上に重ねて書かれる．共通アクセスが必要とされない限り安全なファイル・システムを設計するのは比較的容易である．G. E. システムは，共有と安全性の**両方**を提供する．第1に，いくらかの安全性が通常の合言葉方式によって提供される．合言葉はユーザによってファイルに割当てられてもよい．アクセスはファイル名と共に合言葉を必要とする．第2に，ユーザ

X は，選択した他のユーザに対して，X のファイルへのアクセスの許可を与える（あるいは拒否する）ことができる．4つの型のアクセスが PERMIT コマンドによって指定される．

EXECUTE
READ(EXECUTE を含む)
WRITE(APPEND を含む)
APPEND

PERMIT ステータスは，ユーザ X によってのみ変更されうる．

G. E. ネットワークについての公表された説明書はない．これを書いている時点で，非公式に，それには12台の G. E. 635 計算機が含まれているといわれている．これらのおのおののメモリ・サイクル時間（36 ビット）は1マイクロ秒であり固定小数点 ADD/SUBTRACT/COMPARE 時間は1.9マイクロ秒である．635 システムは割込システム，特権モードおよび再配置用の単一アドレスレジスタ(およびプロテクション)を有している．付加できる装置の中にドラムがある．各ドラムは470万バイト，平均アクセス18ミリ秒，および370キロバイト/秒の転送速度を有している．固定ヘッドおよび置換型ディスクもまた付加できる．これらの装置のいくつかは疑いなくそのシステムに含まれている．

システム構成あるいは性能に関する情報は公表されていない．非公式には，現在，このネットワークは同時に1000のユーザに対して“快適な”応答時間でサービスしているということが知られている．

演　習　問　題

1. (a) タイムシェアリング・システムに独特な主な利点は何か．
(b) バッチ・システムに比較して不利な点は何か．

2. 現代のマルチプログラミング・システムによって必要とされるものに加えて，タイムシェアリング・システムによって必要とされるものとして，次のものを確認しそれについて記述せよ．
(a) CPU 特徴
(b) その他のシステム・ハードウェア

3. スワッピングはタイムシェアリング・システムにおいてしばしば使われ

る方策である.

（a）　それは何か.

（b）　その利点と不利な点は何か.

4.　**専用**と非専用タイムシェアリング・システムの間の差異を論ぜよ.

5.　（a）　ユーザ・コマンドとは何か. バッチ・システムにおいてそれらに対応するものは何か.

（b）　どのような型のコマンドがもっとも必要とされるか.

6.　（a）　タイムシェアリング・システムにおいて, どのような型のプログラム割込機能が特別なコンパイラ編成を要求するか. それらの機能はバッチ・システムでは必要とされない.

（b）　必要とされるコンパイラの機能を簡潔にのべよ.

7.　インタプリタとコンパイラの基本的編成と性能の特性を, おのおのの利点と不利な点を含めて, 対比させよ. おのおのにとっていつバインディング・タイムが起こるか.

8.　（a）　タイムシェアリング・システムに対する2つの適切な性能尺度は何か.

（b）　これらの基準を満足するために, どのようなスケジューリング・アルゴリズムが役に立ちうるか.

9.　（a）　タイムシェアリング・システムにおいてタイム・スライスを選択する際に考慮すべきことを論ぜよ.

（b）　どのような2つの型のタイム・スライスがしばしば用いられるか.

10.　CTSS システムではなぜ同じサイズの2つの記憶装置ユニットが使われたのか.

11.　CTSS で使われた“指数”スケジューリング・アルゴリズムについて述べよ. このアルゴリズムはどのようにして長く走っているトランザクションが締出されるのを防いだか. CTSS は軽微トランザクションをどのようにして識別したか.

12.　APL 言語プロセッサがインタプリタであるという事実から由来する便宜, 利点, および不利な点のいくつかは何か.

13. APL I ビームは次のものに関するデータを得るためにどのように用いられうるか.
 (a) APL トランスレータの性能は？
 (b) ある APL プログラムの性能は？
 (c) APL システムの性能は？

14. 使用開始中の APL ユーザに対して，主記憶装置駐在の端末メッセージ・バッファを用いることの目的は何か.（それに代わるもう1つの方法は，端末のバッファリングを単に各ユーザの作業空間内に含ませることであろう.）

15. タイムシェアリング・システムのための内部的および外部的性能測定技術の間の特性と差異について述べよ.

16. “スヌーパ端末”がタイムシェアリング・システムにおいて性能を測定するためにどのように使われうるかについて述べよ.

17. **スクリプト**とは何か，また，それはタイムシェアリングの性能研究でどのように用いられるのか.

18. (a) タイムシェアリング・システムをスケジューリングする際のアドミッション規則とは何か，そしてそれはどのように適用されるのか.
 (b) 相補置換えとは何か.

19. シミュレータにおいて確信を確立するために何がなされるべきか.

20. **反応時間**という用語を，タイムシェアリング・システムで用いるものとして，定義せよ.

21. TSO 端末ユーザはどのようにして自分自身のコマンドを作り出すことができるか.

22. TSO で何が起こるかを述べよ.
 (a) オペレータがシステム(IPL)を開始するとき.
 (b) 端末ユーザがログオンするとき.

23. TSO における拡張リンク・パック・エリヤに対するモジュールの選択に伴う性能の論点について論ぜよ.

24. TSO スケジューリングにおける主タイム・スライスと副タイム・スラ

イスの機能について述べよ.

25. (a) 長く実行中のトランザクションを扱うために, どのようなTSOスケジューリングの特徴が企画されているか.
 (b) 軽微トランザクションに対して迅速な応答を保証するために, どのようなTSOスケジューリング・パラメータが企画されているか.
 (c) バックグラウンド(バッチ)作業負荷を扱うために, どのようなTSOスケジューリングの特徴が企画されているか.

26. 回線集中装置とは何か, そしてそれらはなぜあるタイムシェアリング・システムで重要なのか.

27. あなた自身が(たとえば)20のタイムシェアリング端末を必要とするある組織の経営者だとみなそう. (たとえば)TSOのようなシステムから利用できるサービスとGEネットワークのようなユーティリティ型システムから利用できるサービスを対比せよ. あなたの議論に含まれるべきいくつかの要因には次のものがある.
 (a) 機能の一般概要
 (b) サービスを当てにできる見込(評価するのは明らかに困難だが, ともかく推測してみよ)
 (c) 使用の容易さ
 (d) ユーザ仕様で作られるサブシステムをインプリメントする能力
 (e) システム・リソースに対する制御の程度
 (f) コスト

■ 28. 5-8節でタイムシェアリング・システムの1つの簡単な数学モデルを展開し, その結果, 式(13)の性能等式, あるいはその回帰形の式(14)および(15)を得た. 図示すると図5-8.1となる. これらを用いて下記に答えよ.
 (a) 40人のユーザが使用開始中として, それらの人の平均思考時間は20秒そして平均CPU時間は0.5秒である. 平均待ち時間はいくらか.
 (b) 上のシステムで, 平均待ち時間が5秒になるまでに何人のユーザが使用開始可能か.
 (c) このモデルは軽微応答時間について何を述べているか.
 (d) タイムシェアリング・システムの性能に影響を及ぼす要因で, そ

のモデルに含まれて**いない**ものをいくつかあげよ.

29. 文献(まず第1に著者目録を見よ)を調べ，次の論題について報告書を書け.「バッチ環境とタイムシェアリング環境におけるプログラマの生産性の比較」(その比較についての自分自身の結論のみならず，使用した方法論の批判も含むこと.)

30. この研究課題はタイムシェアリング・システムが実際に利用できる必要がある. 次のことを含む“スヌーパ端末”の研究を企画し，それを行なえ.
   (a) テスト・スクリプト(軽微および非軽微トランザクションの両者を含む)を選んで定義せよ.
   (b) CPU時間，経過時間，およびプリント時間を測定せよ.
   (c) 1日あるいは数日間にわたる数時間で測定せよ.
   (d) それらのデータを表形式に整理せよ. そしてその結果を分析するために2章の技法を用いよ.
   (e) 調査結果と結論について報告書を書け.

31. 本文中のもの以外のタイムシェアリング・システム1つについて報告せよ. それを本文中で議論されたAPLあるいはTSOと比較せよ.

参考文献

1. [AN] Anderson, H.A., and R.G. Sargent: A Statistical Evaluation of the Scheduler of an Experimental Interactive Computing System, in "Statistical Computer Performance Evaluation," W. Freiberger, ed., Academic Press, New York, 1972.
2. [BA] Bard, Y.: Performance Criteria and Measurement for a Time-Sharing System, *IBM Systems Journal*, vol. 10, no. 3, 1971.
3. [BE] Bernstein, A.J. and J.C. Sharp: A Policy-Driven Scheduler for a Time-Sharing System, *Comm. ACM*, vol. 14, no. 2, February, 1971.
4. [BL] Blatny, J., S.R. Clark, and T.A. Rourke: On The Optimization of Performance of Time-Sharing Systems by Simulation, *Comm. ACM*, vol. 15, no. 6, June, 1972.
5. [BRE] Breed, L.M. and R.H. Lathwell: APL/360, IBM Contributed Program Library Document 360D-03.3.007, 1968.
6. [BRY] Bryan, G.E.: JOSS 20,000 Hours at a Console: A Statistical Survey, *AFIPS Conf. Proc.*, *FJCC*, vol. 31, pp. 769-778, 1967.
7. [CO1] Coffman, E.G. and L. Kleinrock: Computer Scheduling Methods and Their Countermeasures, *AFIPS Proc.*, *SJCC*, vol. 32, 1968.
8. [COR] Corbato, F.J. et al.: An Experimental Time-Sharing System, *AFIPS Conf. Proc.*, *SJCC*, vol. 21, pp. 335-344, 1962.
9. [DO] Doherty, W.J.: Scheduling TSS/360 for Responsiveness, *AFIPS Conf. Proc.*, vol. 37, pp. 97-111, 1970.

10. [GE] General Electric Co.: Mark II Command System Reference Manual, No. 711223E, General Electric Co. Information Services Dept., 7735 Old Georgetown Rd., Bethesda, Maryland 20014.
11. [GO] Gold, M.M.: Time-Sharing and Batch Processing: An Experimental Comparison of Their Values in a Problem Solving Situation, *Comm. ACM*, vol. 12, no. 5, May, 1969.
12. [HE1] Hellerman, H.: A Personalized Array Translator System, *Comm. ACM*, vol. 7, no. 7, July, 1964.
13. [HE2] Hellerman, H.: Some Principles of Time-Sharing Scheduler Strategies, *IBM Systems Journal*, vol. 8, no. 2, 1969.
14. [HE3] Hellerman, H., and Y. Ron: A Time-Sharing System Simulation and Its Validation, *IBM Tech. Report 320-2984, IBM New York Scientific Center*, April, 1970.
15. [HE4] Hellerman, H.: Complementary Replacement–A Meta Scheduling Principle, *Second ACM Symposium on Operating System Principles*, Princeton University, 43-46, October 20-22, 1969.
16. [IB1] IBM Corp: "IBM System/360 Operating System: Time-Sharing Option: Command Language," No. GC28-6732, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
17. [IB2] IBM Corp: "IBM System/360 Operating System: Time-Sharing Option: Terminal User's Guide," No. GC28-6763, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
18. [IB3] IBM Corp: "IBM System/360 Operating System Time-Sharing Option Planning for TSO," No. GC28-6698, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
19. [IV] Iverson, K.E.: "A Programming Language," John Wiley and Sons, New York, 1963.
20. [LA] Lampson, B.W.: A Scheduling Philosophy for Multiprocessing Systems, *Comm. ACM*, vo. II, no. 5, May, 1968.
21. [LE] Levy, D., A. Kellerman, H. Hellerman, and J. Kaufman: MIN/TSO Primer, State University of New York at Binghamton, 1972.
22. [MA] Maranzano, B.J., and B.J. DiMarisco: TSO Performance Analysis Routines User Guide, SHARE Document No. 360D04.4.012.
23. [MA1] Martin, J.: "Design of Real-Time Computer Systems," Prentice-Hall, New York, 1967.
24. [NI1] Nielson, N.R.: The Simulation of Time-Sharing Systems, *Comm. ACM*, vol. 10, no. 7, pp. 397-412, 1967.
25. [NI2] Nielson, N.R.: Computer Simulation of Computer System Performance, *Proc. ACM*, 22nd Natl. Conf., pp. 581-590, August, 1967.
26. [OP] Oppenheimer, G., and N. Weizer: Resource Management for a Medium Scale Time-Sharing System," *Comm. of the ACM 11*, no. 5, pp. 313-322, May, 1968.
27. [PI] Pinkerton, T.B.: Performance Monitoring In a Time-Shared System, *Comm. ACM*, vol. 12, no. 11, November, 1969.
28. [SA1] Sackman, H.: Time-Sharing vs. Batch Processing: The Experimental Evidence, *AFIPS Conf. Proc., SJCC*, vol. 32, pp. 1-10, 1968.
29. [SA2] Sackman, H.: "Man-Computer Problem Solving; Experimental Evaluation of Time-Sharing and Batch Processing," Auerbach Publishing Co., New York, 1970.
30. [SC1] Scherr, A.L.: "An Analysis of Time-Shared Computer Systems," MIT Press, Cambridge, Mass., 1967.
31. [SC2] Scherr, A.L. and D.C. Larkin: Time-Sharing for OS, *AFIPS Proc., FJCC*, vol. 34, 1970.
32. [SC3] Schatzoff, M., R. Tsao, and R. Wiig: An Experimental Comparison of Time-Sharing and Batch Processing, *Comm. ACM*, vol. 10, no. 5, pp. 261-265, May, 1967.

33. [SH] Shemer, J.E. and D.W. Heying: Performance Modelling and Empirical Measurement in a System Designed for Batch and Time-Sharing Users, *AFIPS Conf. Proc.*, vol. 35, pp. 17-26, 1969.
34. [SM] Smith, L.B.: A Comparison of Batch Processing and Instant Turnaround, *Comm. ACM*, vol. 10, no. 8, August, 1967.
35. [ST1] Stimler, S. and K.A. Brons: A Methodology for Calculating and Optimizing Real-Time System Performance, *Comm. ACM*, vol, 11, no. 7, pp. 509-516, July, 1968.
36. [ST2] Stimler, S.: Some Criteria for Time-Sharing System Performance, *Comm. ACM*, vol. 12, no. 1, January, 1969.
37. [STR] Strauss, J.C.: A Simple Thruput and Response Model of EXEC 8 under Swapping Saturation, *AFIPS Conf. Proc.*, vol. 39, pp. 29-49, 1971.
38. [TO] Totaro, J.B.: Real-Time Processing Power: A Standardized Evaluation, *Comput. Autom.*, vol. 17, no. 4, pp. 16-19, April, 1967.
39. [WA] Watson, R.W.: "Time-Sharing System Design Concepts," McGraw-Hill Book Company, New York, 1970.

第 10 章

仮想記憶装置の原理

いずれの計算機システムの設備にも，記憶される情報の単位当りの速度とコストが異なるいくつかの型の記憶装置が含まれている．この多様性は，技術上の1つの不変な事実——安価な記憶装置はおそく，はやい記憶装置は比較的高価である——の結果である．この理由のために，ほとんどの計算機システムは，小さなはやい記憶装置，より大きく・遅く・安い記憶装置，そしてさらにより大きな容量でより安い型の記憶装置，などからなる**記憶装置の階層**で構成されている．簡単でよく知られている例として，主記憶装置と補助記憶装置の2階層構成がある．高速側の構成要素である主記憶装置は現在マイクロ秒領域のアクセス・タイムを持ち，典型的には磁気コアあるいはソリッド・ステートの集積回路のようにすべて電子技術で作られたものである．低速側の構成要素は，典型的には，ミリ秒のアクセス・タイムを持つ磁気ドラムあるいはディスクのように電気-機械構成となっている．主記憶という名は，命令語とそれらが参照するデータが実行の瞬間には主記憶装置の内に存在しなければならないので，そのように呼ばれている．あるプログラムとそのデータによって必要とされる全論理記憶装置は主記憶装置内の利用可能な空間をはるかに越えている．あるプログラムによって必要とされる全記憶装置は補助記憶装置上で供給され，この全体のうちの断片の写しが，おのおのの断片の実行に先だって，補助記憶装置から主記憶装置に移動される．処理された断片は，その結果を保存し，全体の問題の他の断片に処理の順番が回るようにするため，逆の方向に移動される．その断片を選ぶ方策（および関連した操作）のことを**折りたたみ** (folding) と呼んでいるが当を得た言い方であり，それはどの計算機システムでも一通りか2通りの方法で行なっている共通の操作である．速度と性能の観点から，折りたたみ操作には，最も頻繁にアクセスされる命令およびデータ・バイトはそのほとんどの時間主記憶装置を占めているべきである，という原理

が適用されるべきである．

伝統的な(非仮想の)記憶装置システムでは，折りたたみは，各個々の問題プログラマの責任であった．第7章および第8章で議論したように，I/O マクロのようなシステム・プログラムや高レベル言語のコンパイラは，プログラムを折りたたむのに使われる主な技術であるI/O操作の詳細の多くを問題プログラマが知らなくてもよいようにしている．しかしながら，それでもなお問題プログラマは，物理主記憶より大きくてはならないある限定された大きさの主記憶装置を仮定して，自分のプログラムを設計し書かねばならないし，また，そのプログラムにおいて情報がいつどこで(さもなくばどのように)主記憶装置と補助記憶装置の間で移動されることになっているかを指定しなければならない．従来のシステムでプログラマによって行われる折りたたみの例には次のようなものが含まれる．

1. 1つの大きなプログラムを**多数のジョブ・ステップ**に分割する．それらステップは，いずれも利用可能な主記憶空間より大きくない空間で，順次に処理されてゆくものとする．
2. 利用可能な主記憶装置にとってあまりに大きなプログラム(すなわちプロシジァ)に対しては**オーバレイ構造**をつくる．オーバレイはプログラマによって指定される．すなわちプログラマは大きなプログラムを部分に分割し，どの部分が主記憶装置内に同時に駐在すべきかを予め計画する．それからこの構造は適切な制御ステートメントでコンパイラあるいはリンケージ・エディタに対して宣言される．
3. 大きな表のように，主記憶装置に入りきらないデータに対しては，“**投げ出し**(spill)”ファイルすなわち作業ファイルを利用する．
4. 1つのジョブ・ステップから別のジョブ・ステップへデータを受渡すために中介データ・ファイルをつくる．

大規模な折りたたみをしばしば必要とするプログラムの1つの一般的なクラスはコンパイラ，特に近代的な複雑な高級言語のそれである．コンパイラによってなされる論理的な機能は，そのオブジェクト・コードの最適化のためのアルゴリズムを含めて，非常に大きなプログラム(たとえば500kバイト以上)となってしまう．そのような大きなプログラムを合理的な大きさの主記憶装置に適合させるために，コンパイラはいくつかのフェーズに分割される．コンパイルの間，ソース・プログラムを処理するため一度にはただ1つのフェーズしか(あるいは2~3ということもありうるが) 主記憶装置に駐在しない．コンパイ

ラのフェーズ構造はオーバレイ技法の1つの型である．

プログラムを人手で折りたたむのに要するプログラマの時間と労力はかなりのものであり（見積りのためには，*Sayre*［SA 参照］を見よ），そしてそれが，2つあるいはそれ以上の記憶装置の階層の間でデータとプログラムを**自動的に折りたたむ**，**仮想システム**と呼ばれる一種のリソース管理機能の開発の十分な動機となっている．自動折りたたみについての利点は他にもまたあるが，それは次の節で述べられる．

最初の実用的な仮想記憶装置システムであるATLASは，1960年代初期に，イギリスのManchester大学とFerranti社の共同の冒険的事業であった．Burroughs，Xerox（その当時，Scientific Data System），IBM，RCAおよびG.E.（MITと提携で）を含んだ多くの他の会社は，仮想記憶装置システムを1960年代中期に導入した．この時代には，大型のBurroughsシステムがその概念を強力に利用していた．1965年にIBMは仮想記憶装置のための特別なハードウェアを有したシステム/360モデル67，モデル65を発表した．モデル67の初期の仮想記憶装置オペレーティング・システムはTSS（Time Sharing System）と呼ばれた．その名が示すように，TSSは仮想記憶装置のみならずタイムシェアリングを目的として設計された．しかし1章で述べたように，タイムシェアリングと仮想記憶装置とは独立な特性である．大体同じ時期に，G.E.と共同でMITはMultics（Multiplexed Information and Computing System）の開発を発表した．MulticsはG.E.635を特別に一部変更したG.E.（今はHoneywell）645の上で仮想記憶装置を提供する1つのオペレーティング・システムである．もう1つのIBMの労作であるM44システムは，特別に変更した7044計算機を用いて作られた．それは，1966年から1969年の間ニューヨークのヨークタウンにあるIBM Research Labで研究の道具として役に立った。R. A. Nelson，R. W. O'Neill，D. Sayre，F. Gustavson，B. Brawn，L. A. Beladyその他によって指揮されたこのプロジェクトは，営利開発の圧力なしに，かなり体系的な方法で仮想記憶装置（自動折りたたみ）の概念を探究した．M44の概念の多くはIBM Cambridge，Mass. Scientific Centerで開発されたCP-67の中で使われた．この仮想機械-オペレーティング・システムは，多くのモデル67ユーザを獲得した．これは1972年に発表されたIBMのVM/370システムの直接の前身である．いくつかの成功を達成したもう1つのモデル67仮想システムは，Michigan大学のMTSシステムであった．

1960年代に得られた仮想記憶装置システムについての経験は，その概念の

実用性と価値を確立した．これは1970年代のさらにより多様な仮想記憶装置システムの開発をもたらした．

10-1　仮想記憶装置機構に対する理論的根拠［BA 2, BE 3, FO, HE, KA, RA 1, SA, SI 参照］

仮想記憶装置の概念に対してまず厳正な検討をすることからはじめよう．たしかに，ユーザ指定によるプログラムの折りたたみは，少なくとも，自動折りたたみ機構にまさる潜在的な効率上の利点を持っている．多分，ユーザは前もって自分の問題の記憶装置アクセス特性の多くを知っていて，したがって，効率のよい折りたたみを指定することができるはずである．しかしながら，プログラマ指定による折りたたみは，過度のプログラミングとデバッグの時間を要求することもあり，また必ずしも実際のデータ依存の記憶装置アクセス・パターンを反映していないかもしれない．特定のユーザの要求に応じてというよりむしろほとんど自動的に折りたたみを行なうということは，長く何人かの計算機システム設計者のビジョンであった．**仮想記憶装置**システムは，通常2つのレベルの記憶装置の間で折りたたみがシステムによって自動的に行なわれる記憶装置の階層から成っており，したがっていかなる特別なユーザの工夫やプログラミングも必要としない．その代り，実際の実行時の記憶装置の要求に応じて，機械論理およびシステム・プログラムが折りたたみ操作の時機と種類を識別する．リソース管理の仮想記憶装置方式の目標は，あらかじめ計画された，物理的な記憶装置の大きさに依存している状態から，プログラムを解放することにある．したがって次のような利点が保証できる．

1. プログラマは手で折りたたみを指定するという困難な仕事から免かれる．
2. たとえ主記憶装置の大きさが縮小されても，あるいはプログラムがより小さなシステムに移されても，そのプログラムは正しく実行されるであろう．しかし，小さな主記憶での実行速度は，かなりおそくなるであろう．逆に記憶装置が増加すれば，そのプログラムは再プログラミングしなくともより速く実行することも多いであろう。
3. マルチプログラミングの環境の下では，オペレーティング・システムは，貴重な高速記憶装置リソースをプログラムに割当てる際，ずっと大きな柔軟性を持つことになる．というのは，それはどのプログラムにもある固定量の記憶装置を供給する必要がないからである．オペレーティング・

システムにとって，主記憶装置は，固定した大きさであり種々の問題の大きさをそれに適合させなければならない空間であるというより，むしろ割当てられるべき空間の**たまり場**と考えてしまってもよいのである．

4. 仮想記憶装置システムは，その高速記憶装置を，プログラム実行に先だってなされたユーザによる最悪の場合の見積りに従ってというより，むしろ**実際に看取される**空間要求に従って割当てる．

この最後の点は特に強調するのに値する．書かれたプログラムおよびそのデータは，プログラムのアドレス空間として7章で述べた“可能な”記憶装置の参照域の1つの集まりを表現している．そのアドレス空間内の多くの番地は，プログラムのある1回の実行の間，参照されないかあるいはめったに参照されない．これは，一部には，あるプログラムの部分を回避するデータ依存の分岐のためである．たとえば，多くのプログラムは3つの部分，すなわち初期化ルーチン，主要ライン・ルーチン，および例外ルーチンから成っている．初期化ルーチンは1回のランにつきただ1回参照されればよい．いくつかの例外ルーチンはあるデータの集まりに対しては決して参照されないかもしれない．実際に参照される記憶装置の番地でさえ一様な頻度で参照されない．記憶装置の同一領域への参照の時間分布もまた大きく変化しうる．典型的には，プログラムの参照は，かなり長い時間にわたって比較的小さな領域にとどまる傾向にある．この性質のことを**参照の局所性**と呼ぶ．もっとも簡単な例はプログラムのループ内の命令語に対する記憶装置の参照である．よく設計された仮想記憶装置システムは高い局所性を効果的に識別し，そのような領域のみを高速記憶装置内に維持する．プログラムの活動がアドレス空間のいろいろな区域に移動するにつれ，これらは主記憶装置内に自動的に入ってくる．

これまで，仮想記憶装置に対する動機のいくつかの概略を述べてきたので，仮想記憶装置管理の基礎理論を形成するいくつかの定義，論理的機能，および記憶装置参照の調査の方法のいくつかについて話を進めてゆく準備ができた．いくつかの例題システムが考えられる．これらの例は2つの種類の**環境**(context)に属することがわかるであろう．(1) 主記憶/補助記憶の階層構造すなわち**システム設計環境**および (2) バッファ記憶/バッキング記憶の階層構造すなわち**CPU 設計環境**である．

システム設計環境は，仮想記憶装置の理論的根拠を議論するのに使われたものである．システム設計環境のための第1の動機は大きなアドレス空間をおのおのの実行ユーザに対して提供することである．それはハードウェアとソフトウェアの技術の結合によって実施される．

CPU設計環境は高速中央処理装置の設計に関連している．この型の階層構造の小さな高速要素(**バッファ記憶**と呼ばれる)は，処理装置とこの階層構造のより低速の要素である大きい主記憶(また**バッキング記憶**と呼ばれる)との間にさしはさまれる．多くの点で，そのバッファは伝統的なシステムにおける主記憶の役割を果たしている．すなわち，それは処理装置のためのデータと命令語の直接の源である．しかしながら，1つの重要な差異がある．ラン中のプログラムは(そのバッファではなく)主記憶装置を参照しつつあるとみなされており，したがって，主記憶とバッファの間のすべての折りたたみ操作はハードウェア回路によって自動的に行なわれ，プログラマには全く見えない．最初のCPU設計環境のシステムはIBM S/360モデル85であった．同様の原理は，後のIBM S/370モデル155, 165, 158および168(あとの2つの機械はまたシステム設計環境をインプリメントしている)に使われている．

仮想記憶装置の2つの環境の間の1つの重要な実際的差異は，物理記憶の速度比である．典型的には，システム設計環境では，その比は5000：1の程度であるが，一方CPU設計環境では，それはほぼ5：1である．予期できるように，このことの結果実施上の細目が全く異なったものとなる．システム設計環境では，自動折りたたみのためのもっとも頻繁な過程はハードウェア回路によって行われるが，その他の過程はオペレーティング・システム・プログラムによって行なわれる．CPU設計環境では，高速の要求はすべてハードウェアで実施される．

次の節では，仮想記憶装置を実施するのに用いられる過程(アルゴリズム)に注意を集中する．具体的な方法でその原理をありありと示すために，特に触れない限り主としてシステム設計環境を用いる．

10-2　アドレス空間とマッピング：ページ方式システム [DE 2, KU 2, LI 1 参照]

プログラムに供給される論理記憶装置空間はそのアドレス空間(AS)と呼ばれ，その概念は第7章で導入された．アドレス空間をありありとした形で示すためには，それを大きな補助記憶装置の連続した領域と考えてもよい(後にわかるように，実際は物理的にこうではないのだが)．非仮想記憶装置システムでは，1つの問題のアドレス空間はその計算機の主記憶装置の大きさより大きくならないように制限されている．この制限は仮想記憶装置システムでは取除かれている．

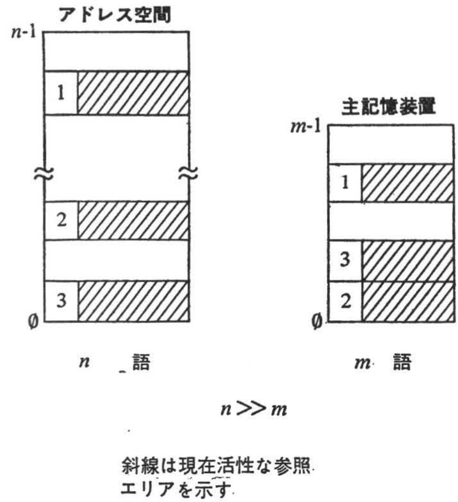


図 10-2.1　アドレス空間と主記憶装置

$n =$ AS の語数

$m =$ 主記憶装置(MS)の語数

としよう．そのとき，

$$n > m$$

が許される．したがって**図 10-2.1** に示されるように，一度には，アドレス空間の小部分のみしか主記憶装置に当てはめることができない．

さて，あるアドレス空間のために書かれた1つのプログラムを想像しよう．その実行中において，その局所性は良好であるが，たびたびアクセスされる命令とデータが少量の空間しか必要としないにもかかわらず連続して存在していないということがよく起こる．すなわち，それらが AS 内のいくつかの(しかし散らばった)**区画**に置かれていることがある．たとえば，数千の命令の実行に際して，命令語の取出しは1つの領域内に納まっているが，データの参照については別の領域に納まっていることがある．このことは，アドレス空間のいくつもの**非隣接**部分が同時に主記憶装置を占有できるということが有利であるということ，を示している．けれども，この自動折りたたみの様子は，その他のすべての様子と同様，ユーザには透明で見えないものでなければならない．仮想記憶装置には2つの基本的な過程が伴っている．

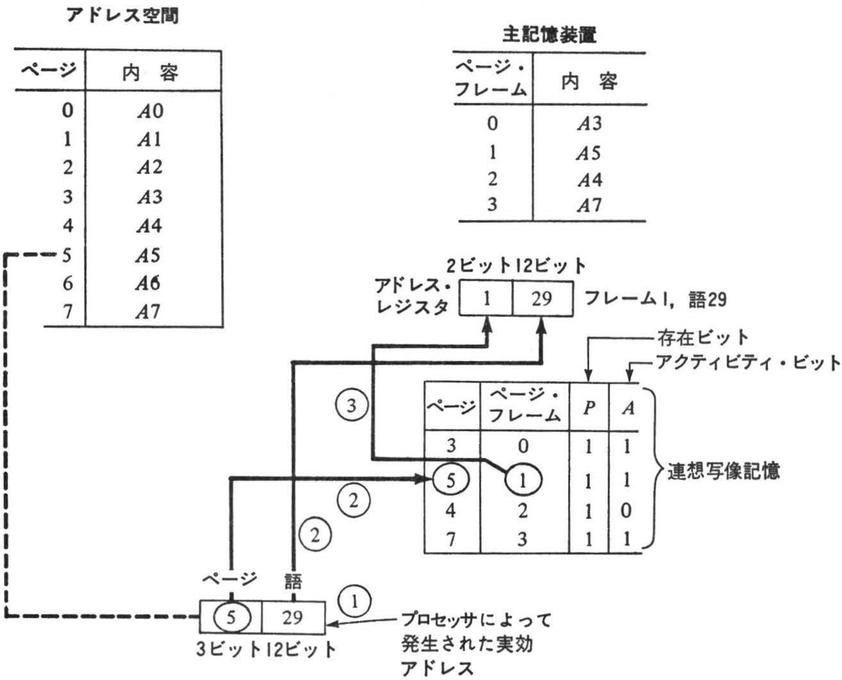
1. 各実行期間中に主記憶装置内に存在すべき AS の部分を**選択すること**．

2. プログラムによって発生された各アドレス(それは AS に対するあるアドレスであるが)を主記憶装置に対する物理アドレスに**写像すること**，すなわち翻訳すること．

ここで，写像機能について少し詳しく議論しよう．**図 10-2.2** は，最も単純化した概念上の写像機構(しかし，必ずしも最も経済的なものではないが)を示している．この機構を**連想写像**と呼ぶことにしよう．$2^{15} = 32\text{k}$ 語，したがってそれをアドレスするために 15 ビットを必要とするアドレス空間が，図に描かれている．この AS は 1 ページあたり 4096 語の 8 ページとして構成されている．それに対するアドレスは“実効アドレス”と名付けられて図に示されているが，これは 3 ビットのページ・アドレスと 12 ビットの語アドレスから成っているものとみなされている．またこの例では主記憶装置は $2^{14} = 16\text{k}$ 語であり，**ページ・フレーム**と呼ばれる(各 4096 語の) 4 ブロックで構成されているとみなされている．(このページ・フレームの用語は，AS のページと主記憶のページの混同を避けるためで，後者をページ・フレームと名付けている．)図 10-2.2 における $A0$, $A1$ などのような値は，システムがしばらくの間ランした後のある瞬間におけるシステムの状態のスナップショットとみなしてよく，その値はそれ以前の活動の結果を示す．現在，ページ $A3$, $A5$, $A4$ および $A7$ が主記憶装置ページ・フレーム内に駐在しているのがわかる．

AS に対する 15 ビット実効アドレスは，主記憶装置内に存在していることを期待しているある 1 語を参照している．この語はどこにあるのであろうか．もっと正確にいえば，ハードウェアはどのようにしてこの語を捜し出すことができるのであろうか．示された例では，それは AS 内の 5 ページのアドレス 29 にある語であることがわかる．さて，われわれの仮想記憶装置をインプリメントしているシステム・プログラムは，すでに，主記憶内のページ・フレーム 1 に AS のページ 5 を配置していると仮定しよう．そして，さらに図 10-2.2 に示されているように，これら 2 つのページ・アドレスの間の**対応** $(5 \rightarrow 1)$ を**写像記憶**と呼ばれる特殊記憶装置のある 1 語内に記録していると仮定しよう．実際，写像装置は，AS → ページ・フレーム・アドレスの対応**テーブル**(表)を保持している．このことがわかれば，図 10-2.2 に示されている写像過程を理解することは困難でないはずである．

示されている写像装置は，ページ・フレームの数と同数の語を持たなければならない．おのおのの写像エントリ(記載事項)は，3 種の情報を保持している．



操作の要約

(1)　プロセッサ実効アドレスはアドレス空間(仮想記憶装置)におけるロケーションを指定する．

(2)　実効アドレスの各部：
- **ページ部**：　写像記憶(左欄)を探索するために使われる．
  - 合うものが見つかる：　第2欄がアドレス・レジスタへ送られる．
  - 合うものがない：　ページ・フォールト；ページ・スーパバイザ・プログラムへの割込．
- **語　　部**：アドレス・レジスタへ送られる．

図 10-2.2　連想記憶による写像

1.　アドレス空間のページ・アドレス．
2.　そのページの主記憶装置アドレス(ページ・フレーム・アドレス)．
3.　制御ビット．すなわち，そのセルが実際の写像データ(あるいは，以前の活動によるゴミ)を含んでいるかどうかを指定するマップ存在ビット(P)や，最近の過去におけるページ活動度のある尺度を与える1つ以上のビット，など．

実際の写像過程は，マップの最左列に対する探索引数として実効アドレスの

3ビット・ページ・アドレスの部分を用いること，および，同時に，マップ・エントリの存在ビット内で1をチェックすること，から成る．対応するものが見つかれば，その写像エントリの主記憶アドレスの部分におけるビットは，主記憶装置内のページ・フレームのアドレスである．そのページ内の語のアドレスは，実効アドレスの低次のビットから直接取り出される．写像装置での探索で対応するものが見つからない場合には，プロセッサは現在主記憶装置に存在しないページを参照しようとしている．このことを**ページ・フォールト**と呼んでいる．写像操作は非常に頻繁に行なわれるので(それらは記憶装置参照のたびに必要とされる)，写像は高速に実行される機能でなければならない．さもないとこの過程はシステムを著しく遅いものにしてしまう．一方，ページ・フォールトは比較的まれであると期待できるが，それらに対する応答はかなり複雑である．それゆえ，その応答は，ページ・フォールトが起こった時に自動的に生成されるプログラム割込によって呼びだされる，オペレーティング・システム・プログラムによって取扱われる．それから，このプログラムは参照されたページを主記憶装置内のどこに置くべきかを決定しなければならない(後の参照のために用いられる目的で保存されている，ある存在しているページを，おそらく置換えることになろう)．オペレーティング・システムはそのページを移動し，古いページ・アドレスを抹消し新しいページ・アドレスを含むように写像装置の内容——写像ディレクトリ——を更新しなければならない．この全過程は**ページ置換え**あるいは**ページめくり**と呼ばれ，これを行なうための数多くのアルゴリズムがある．そのうちのいくつかを後で議論する．

写像は，写像ディレクトリを探索するために実効アドレスの高次ビットを用いることによって，行なわれると仮定されてきた．この探索はどのようにして成し遂げられるのであろうか．線型探索は，たとえ高速の写像記憶であっても，耐え難いほど遅い．2分探索は実際的ではない．というのは，これには順次リストが必要で，また，この順序付けのためにかなりの処理時間を要する(さらに，その時でさえ，1回の探索に数回のアクセスを要する)からである．高速写像するにはハードウェアによって解決することが望ましい．写像装置が図10-2.2に示されているように最小数のセルを使用するとすれば，それは，通常，**連想記憶装置**，すなわち高速**並列探索**の可能なもの，である．このことは，それが内部論理回路を持っていることを示しており，それによって，ある与えられた引数とその語の**すべて**の間での探索が1回で(並列に)なされうる．ページの大きさを選択するにあたって1つ考慮すべきことは，その結果で決まる写像記憶装置の大きさである．もしそれが連想記憶で実現されるとすれば，

通常の記憶装置よりも記憶ビットあたりかなり高価なものになるからである．特にある与えられた主記憶の大きさに対して，ページの大きさを小さくすればするほど，ページ・フレームの数は多くなり，必要とする写像記憶装置は大きくなる．逆の言い方をすれば，ページの大きさを大きくすると写像ハードウェアは経済的になる．また補助記憶装置と主記憶装置の間で小さなブロックより大きなブロックを転送する方が，1語あたりの移動にかかる時間においても，経済的になる傾向がある．これらの利点は，ページの大きさを大きくすればするほど主記憶内に同時に保持できるページ(したがって非隣接領域)の数が少なくなるという事実によって，少なくとも部分的には相殺される．これらの矛盾する要求の間の最良の折衷案は個々のシステムの多くの因子に依存しており，通常，システム開発時におけるシミュレーション研究によって解決される．

図 10-2.3 は，別の型の写像機構であり，これは**アドレス方式写像**と呼ばれる．その写像装置は標準の記憶装置編成を使用しているが，AS 内のページの数と同数のセルを必要とする．実効ページ・アドレスは単純に写像ディレクトリへのアドレスとして用いられ，その語内のマップ存在ビットが1であることがわかった場合には，その写像セルは，参照されているページの，求めたい主記憶装置内での位置を(その他の情報と共に)保持している．通常編成(非連想)型の写像記憶装置の経済性は，AS 内のページ数と同数のエントリを有する写像ディレクトリを必要とするという浪費を相殺して余りある場合もある．これは，普通，ページ・フレーム数よりもはるかに多くのエントリとなる．(先に説明した連想機構では，ページ・フレーム数のエントリしか必要としない．)比較的小さなアドレス空間，および写像セルあたり少量の写像情報しか必要としない簡単なシステムに対しては，アドレス方式写像はまったく実現可能である．

写像ディレクトリの内容の制御は，主記憶装置，すなわちマルチプログラミングあるいはタイムシェアード・システムの死命を制する共有リソース，の制御を意味する．この理由のために，写像ディレクトリ内の変更はオペレーティング・システムのみによってなされ，決して問題プログラムによってはなされない．典型的にいって，オペレーティング・システムは，次の時には必ず，ディレクトリの内容(あるいはディレクトリへの1つ以上のポインタ)を変更しなければならない．(1) 1つのページが主記憶内に入れられたとき，あるいは置換えられたとき(この場合，ディレクトリ・エントリは変更されなければならない)．(2) プロセサが1つのプログラムから他のプログラムに切換えたとき(この場合には，普通ディレクトリに対するポインタが変更される)．

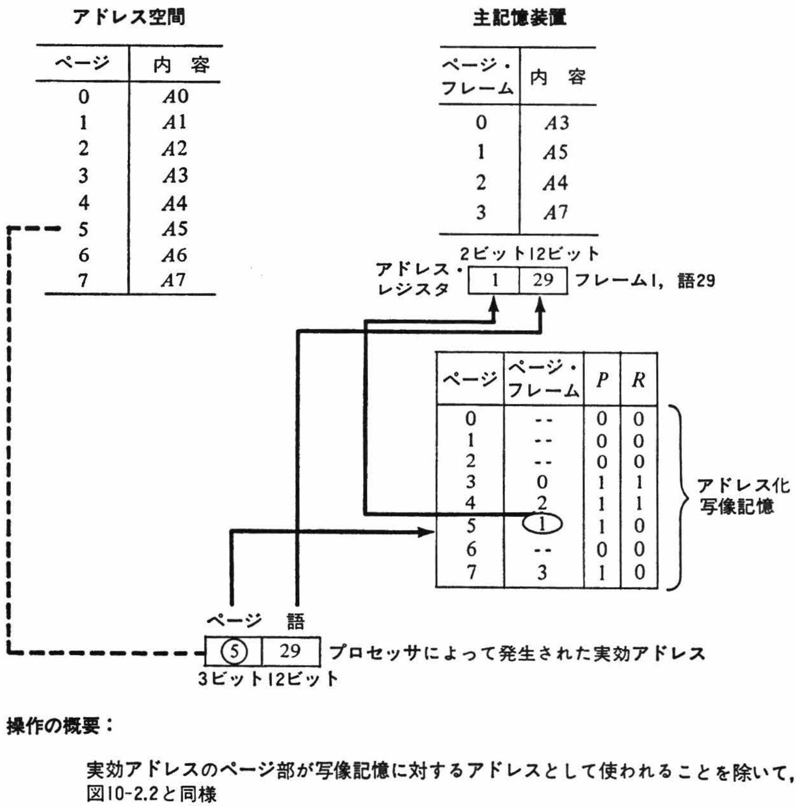


図 10-2.3　アドレス方式主記憶装置写像装置による写像

10-3　セグメンティション[BA1, BU, DE4, GR, KE2, RA2参照]

現代のプログラミング・システムによれば，少なくとも次の理由のために，プログラムは，普通，論理ブロック構造を持つことになる．

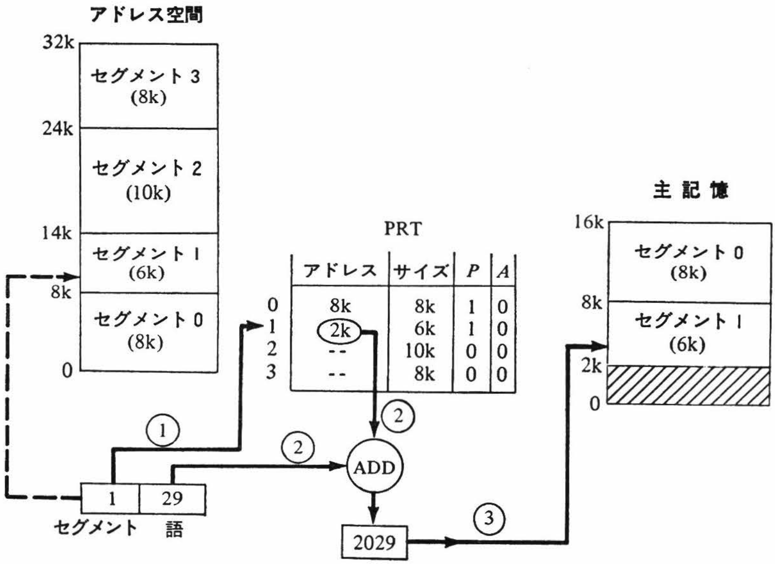
1. プログラムのモジュラリティ．各モジュールには1つの名前が与えられる．モジュールは，そのモジュール内での名前に対する1つの範囲(スコープ)を意味している．各モジュールは別々にコンパイルされてもよい．
2. データ構造．例えば，配列とかリスト．
3. 認められていないアクセスに対する保護．
4. 共有．プログラム A がすでに他の目的のために使用している B 内でモ

ジュール5がアドレスされてしまっていても，AはBから5を借りることを望んでもよい．

セグメント方式アドレス空間は全アドレス空間を，そのおのおのが隣接した領域である多くのセグメントに分割されたものと見なす．Burroughs社のB 5500のようないくつかの仮想記憶装置システムは，セグメントのみを有しページング構造を有していない．IBMモデル67，158および168のような他のシステムは，整数個のページから成るセグメントを有している．いずれの場合においても，1つのモジュールあるいはデータ構造は1つのセグメントに割当てることができるので，1番目と2番目の性質を満足している．保護情報をセグメントに関連させることによって，3番目の性質は満足される．各ユーザのアドレス空間内の唯一のセグメントに共有モジュールを置くことによって，単一の物理コピーが共有されうる（4番目の性質）．

論理セグメントという用語は，プログラム・モジュールあるいはデータ構造のような，プログラマあるいはコンパイラによって名前を付けられた記憶装置の隣接領域を示す．したがって，1つの完全なプログラムおよびそのデータは，論理セグメントの連結した一つの集合である．それゆえ，1つのアドレス空間は論理セグメントに分割される．プログラムとデータをこのように自然な形で区分することが，前に述べたページ方式システムとまったく異なった型の仮想記憶装置の基礎になっている．Burroughs社のB5000，B5500，およびB6500コンピュータがその例である．われわれの議論は特にB5500と関連がある．そのプログラム・アドレスは，**図10-3.1**に示されるように，プログラム・リファレンス・テーブル（PRT）と呼ばれる主記憶装置の1領域を直接参照する．PRTの各エントリは，**記述子**と呼ばれる48ビットの一語で，それには4つの型がある（個々の型は，記述子語自身内のビットによって規定される）．1つの記述子の型は，単に，データである．別の型は，1つのセグメントを記述しているが，実際には，これは個々のセグメントに対して変換マップ内にそのエントリを入れることによって行なわれる．それは，そのセグメントの主記憶装置アドレス，そのセグメントの大きさ，およびそれが主記憶装置内に存在しているかどうかを示す存在ビットを規定している．

もしあるセグメントの外側にアドレスされようとすると，モニタ回路がサイズ違反を検出し，オペレーティング・システムへのプログラム割込みが起こる．なされようとした参照がサイズ・テストを通過しても，そのセグメントが主記憶装置内に現在存在していないときは，そのことが存在ビットのテストの失敗によって発見される．そしてその参照しようとしたことが，結局は，セグ



注：　1．プログラム参照テーブル(PRT)は，セグメントを主記憶アドレスに写像する．セグメントを境界外参照から保護する(サイズ・フィールド)．
2．各プログラムはそれ自身のPRTを持つ．
3．PRTは主記憶装置内に保持される．しかしまた小型連想記憶をもっとも活性な参照の高速写像に用いてもよい．

図 10-3.1　セグメント方式アドレス空間におけるアドレス翻訳

メント-フォールト割込みを引き起すことになる．これはそのセグメントの主記憶装置への移動を要求する，オペレーティング・システム内の，あるプログラムを起動する．

各プログラムはそれ自身の PRT を持っている．あるプログラムが別のプログラムの空間へアクセスしないようにするための記憶装置保護は，現在実行中のプログラムの PRT の始点を指し示すレジスタへのセットはオペレーティング・システム(特権モード命令)にしか出来ないようにすることによって，部分的に保証できる．同様に，オペレーティング・システムのみが，記述子のアドレスおよびサイズ・フィールドを変更出来るのである．論理的には PRT は主記憶装置内に保持されるのであるが，いくつかのモデルでは，小数の高速レジスタあるいは小さな高速メモリが最も現在活性な記述子を保持するために使われている．これは写像過程の速度をかなり上げる．

セグメント方式仮想記憶装置とページ方式のそれとの性質を対比するのは興

味がある．どちらの場合も，論理アドレス空間は主記憶装置より大きい．ページ方式システムにおいては，アドレス空間は，普通，問題プログラマには透けて見える同じ大きさのページに分割される．セグメント方式システムにおいては，アドレス空間は可変長のセグメントに分割され，それらのおのおのは，名前の付けられたプログラム・モジュールやデータ配列のように，問題ごとの意味を持つことが多い．しかしながら，Burroughs システムにおけるごとく，オペレーティング・システムは，行列のような1つの対象を自動的に数多くの部分に分割し，各部分に対して1つのセグメントを創成してもよい．各セグメントは記憶装置を効率的に利用する傾向がある．というのは，その大きさは，通常，機能的な問題ごとの記憶装置単位に対応しているからである．ページは，固定の大きさであるので，使用されない部分を持つことがよく見受けられるが，この現象を**内的フラグメンティション**(破砕)と呼ぶ．一方，記憶装置の効率的な割当はページ方式アドレス空間の方がずっと容易である．というのは一様な大きさの構成単位が常に割当てられ，置き換えられているからである．セグメント方式システムでは，可変の大きさの記憶装置ブロックが割当てられねばならない．この過程では，使用されない空間の“穴”を残しがちである．この穴は，寄せ集めないままでいると，大きな単一セグメントの要求を満足するために利用することができない．これは**外的フラグメンティション**と呼ばれる．1つの救済法は時々セグメントを移動し，穴を寄せ集めること，すなわちすべての未使用空間を隣接させることである．これはかなりのオーバヘッドという不利を招くことになりうる．情報移動という単純な穴寄せ手法の他にも，いろいろな配置および記憶装置管理機構が提案されている(DE2 および KN を参照せよ)．一般的にいって，その問題はリスト処理システムにおける“**ごみ集め**(garbage collection)”に非常に似ている．セグメント方式アドレス空間のための写像過程は効率的となる傾向がある．というのは，写像された対象が機能上識別できる問題あるいはデータのセグメントであるからである．これらは，普通，特に大きなプログラムでは，必要とされるであろうページ数より少ない．

多くの点で，ページ方式機構とセグメント方式機構とは相補的な利点と欠点を持つ．それゆえ，両者の利点を結合しようと努めるのは当然である．**図 10-3.2** は，IBM モデル 67 計算機で使われているものに非常に似た，特定のセグメント・ページ方式システムに対する写像過程を示している．AS の構造は，実効アドレスの次のような分割方法によって示されている．

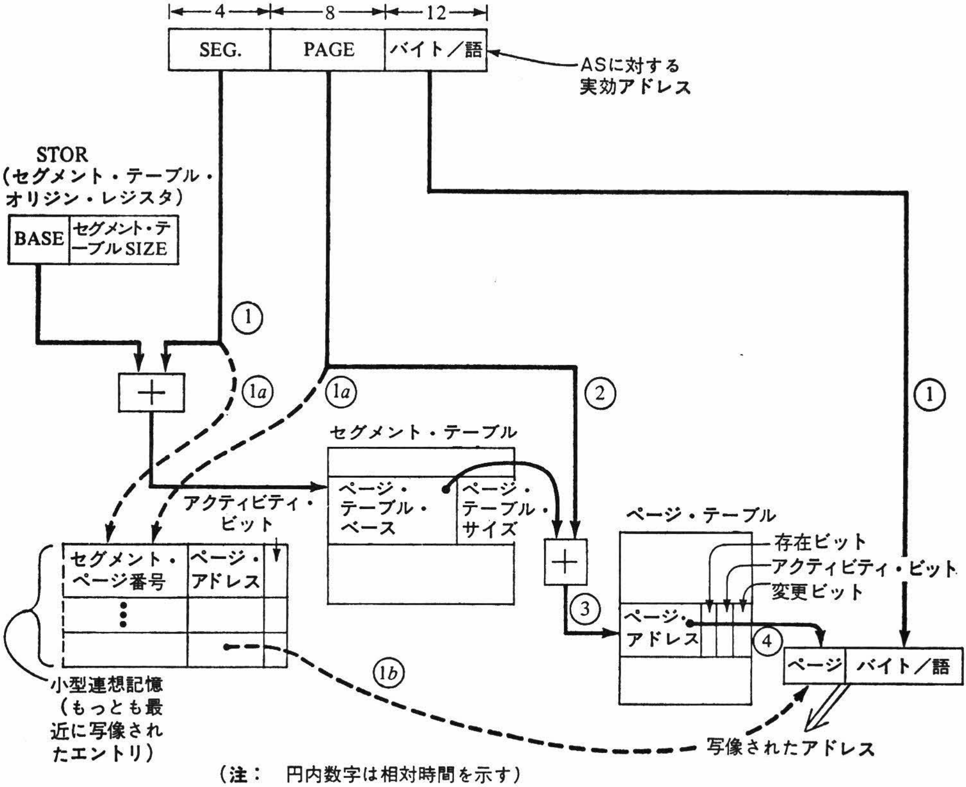


図 10-3.2　セグメント・ページ方式仮想記憶装置システムに対する写像

16セグメントの1つを示すための4ビット
1セグメント当たり256ページの1つを示すための8ビット
1ページ当たり4096(＝4k)バイトの1つを示すための12ビット

それゆえ，24ビット・アドレスは$2^{24}=16$メガバイトのアドレス空間(これはまた**仮想記憶装置サイズ**と呼ばれる)を可能にしている．写像のための設備と過程が図10-3.2に示されている．(処理装置によって発生された)ASに対する実効アドレスから主記憶装置に対する写像物理アドレスへの変換は，主記憶装置内に保持された2つの表といくつかのレジスタによって行なわれる．すべてのレジスタと表は(オペレーティング・システム・プログラムによってあるいはハードウェアによって自動的に)あらかじめ割当てられた値を持ってい

ると仮定する． 読者は図を参照して言葉による記述をたどっていただきたい(円内の数字は事象の時間的順序を示す)． さしあたりは， 破線を無視すること，それは後に述べるように高速化の技法である．写像過程は，実効アドレスの4ビットSEG部にセグメント・テーブル・オリジン・レジスタ(STOR)のBASE部を加えることから始まる． このレジスタはセグメント写像テーブルの始点を指し示しているので，その和はセグメント・テーブル内の1つのエントリを指し示している．セグメント・テーブル内の主なエントリはそのセグメントを記述しているページ写像テーブルのベース(すなわち始点)に対するポインタである． それから， そのページ・テーブル・ベースは実効アドレスのPAGEビットに加えられるが， その和がページ・テーブルへのポインタになる． そこで見つけられる値が写像ページ・アドレス(ページ・フレームを指している)であるが，これが実効アドレスの12ビットのバイト部と連接され最終の物理写像アドレスとなる．

以上，写像過程の最重要点を簡潔に述べた．ここで省略されていたいくつかの重要な詳細について考察しよう．各ページ・テーブル・エントリ内の制御ビットに注目しよう．

存在ビットは，そのページが主記憶装置内に存在するかどうかを指定する．

アクティビティ・ビットは，そのページに対する最近の参照活動のある尺度を与える．

変更ビットは，そのページが最後に主記憶装置内に移動されて以来，そのページの内容に対して変更がなされたかどうかを指定する．

存在(presence)ビットは参照のたびに1かどうかテストされ， 1でないことがわかった場合には，そのページは主記憶装置内に入っていないことを意味する．その時，**ページ・フォールト**割込みが発生する．**アクティビティ**(activity)ビットは，説明が最も入り組んでいる．今のところ，その目的はオペレーティング・システムが(主記憶装置が一杯でページ・フォールトが起こったとき)置き換るべきページとして良いものを選べるように手助けすることにあることに注目しよう．それには，最近の過去において，ともかく比較的不活性であったページを選ぶのである(これについては後に詳しく述べる)．**変更**(change)ビットは置き換え操作の間，ページ転送数を縮少するのに有益である．未変更のページが置き換えられる場合には，その写しをとっておく必要がない．現在の写

しはすでにディスクあるいはドラムに存在している．

図10-3.2で注目に値するもう1つ別の特徴は，STORおよびセグメント・テーブル・サイズ内のSIZEエントリである．それらは参照されているテーブル内のエントリの総数を規定している．これによると主記憶装置は完全なテーブルを保持する必要はなく，不完全なテーブル(これが普通である)を安全に利用することができる．もしそれがインプリメントされると，この特徴によって写像テーブルに対して必要な主記憶装置の量はかなり縮少される．

セグメント・ページ方式システムにおいて写像のために使われるテーブルは，普通，主記憶装置内にある．したがって，プログラムの記憶装置への参照のおのおのは，3回の主記憶装置への参照を必要とするように思われる．すなわちセグメント・テーブルに対して1回，ページ・テーブルに対して1回，そして最後にデータに対して1回である．このことは，1回のみの参照しか必要としない通常のシステムと比較して，ほとんどの仮想記憶装置への参照の速度を非常に遅いものにしてしまうことになるだろう．この速度の不利を回避するために，高速の連想記憶がアドレス写像と共に用いられる．小型で非常に高速の連想記憶(IBMモデル67では8セル)が最も最近に参照されたテーブル・エントリを保持するために用意される．したがって，図10-3.2で破線で示されたように，写像過程はまず連想探索から試みられる．もしそれが成功すれば，写像遅れはほとんどない．もし成功しなければ，その時は上に述べられたより遅いテーブル・アクセシング過程が次に続いて行なわれ，その結果が連想記憶内の1つのエントリに取って代わる．いずれのエントリが置き換えられるのか？　普通使われるアルゴリズムは次の様である．連想写像記憶内の各エントリは，1つのアクティビティ(活動度)・ビットを持っている．これらのすべてのアクティビティ・ビットは初めに0にセットされている．セルが参照されるたびに，そのアクティビティ・ビットは1にセットされる．**すべて**のアクティビティ・ビットが1になると，それらはすべて0にセットされる(というのは，そのときシステムはもはや相対的な活動度を区別することができないからである)．連想記憶内の1つのセルの置き換えが要求されたときに，0アクティビティ・ビットを持った最初のセルが置き換えられるべきものである．

セグメント・ページ方式システムにおける記憶装置保護は，オペレーティング・システム**のみ**にSTORレジスタおよびページ・セグメント・テーブルを変更させる(そして決して問題プログラムにはさせない)ことによって達成される．セグメントはプログラムの間で共有されてもよいが，それはそのような各プログラムのセグメント・テーブルに同一のページ・テーブルを参照させるこ

とによって行なわれる.

前書きでも述べたように，大きなアドレス空間あるいは仮想記憶装置は，小さな高速の記憶装置に**自動的**に折りたたまれる．これまでわれわれの例では，仮想記憶装置，すなわち補助記憶装置と主記憶装置の間のシステム設計環境における折りたたみについて述べられてきた．折りたたみ過程の写像（あるいは翻訳）と主記憶装置管理の必要条件について注意を集中してきた．後の節では，仮想記憶装置システムを実現するための補助記憶装置管理の要求条件について述べる．けれども，次の節ではプログラムの参照の局所性を決定するための1つの手段——仮想記憶装置の実際性に対する理論的根拠に焦点を合わせよう．

10-4　アドレス・ストリームの捕捉

仮想記憶装置の成功は，プログラム実行中における論理記憶装置（すなわちアドレス空間）に対する参照のパタンの性質にかかっている．これらのパタンの理解が仮想記憶装置システムの合理的な設計にとってきわめて重大である．そのような理解は，作業負荷(大きな物理記憶を使う)を代表する数多くの標本プログラムを実際に実行し，そのようなプログラムを，**毎回の**記憶装置への参照の記録で，監視することによって得られる．実行の結果得られる記憶装置へのアドレスの系列は，**アドレス・ストリーム**(あるいは，アドレス・トレース)と呼ばれ，通常磁気テープ上に格納される．ここで，アドレス・ストリーム情報の特性を記述する方法を手短かに議論しよう．けれども，順序としてまずアドレス・ストリームがどのようにして得られるのかについて，またそれらの性質についていくつかの観察をする．

アドレスは主記憶装置アドレス・レジスタに現われるので，それらを捕捉するハードウェア・モニタリング装置を想像するのは容易であるが，実際にはこれは普通使用されている手段ではない．まず，すぐ後でわかるように，たとえ中型計算機であってもアドレス発生率は非常に高いので，そのままの発生率の情報を受理できる適度に大型で経済的な記憶装置がない．また，アドレスのみならずそのソース(プログラムあるいはデータ)および型(フェッチあるいはストア)のようなその他の情報を参照のごとに記録するのが普通望ましい．これらの理由のために，望みのプログラムをインタプリティブに実行することによって，そのアドレス・ストリームを捕捉することが通例である．それゆえ，少なくともその計算機のアドレス決定部分をシミュレートするインタプリタ・プログラムが設計される．標本プログラムは，このインタプリタのもとで実行する．インタプリタは各実効アドレスを計算し，CPU に対してそれを記憶装置

参照のために使用することを許可する前に，ストリーム内にそれを記録する．良く知られているように，インタプリタはプログラムの実行をかなり遅くする(10から100倍に遅くなることは珍しくない)．したがって，アドレス・ストリームを得るのは時間的に高くつく．時間のコストは，アドレスとそれに密接に関係した情報のみを記録することによって，出来るだけ切り詰める．そのストリームに基づくデータ整理とシミュレーションは別の計算機を走らして行なわれる．

簡単な計算をすると，1つのアドレス・ストリームにおける大量の情報に対して健全な尊敬の念を払うべきことがわかる．中型計算機を考え，その命令速度をかりに毎秒100万命令としよう。ほとんどの命令は2つのアドレス(その命令に対して1つ，データに対して1つ)を発生する．1つのアドレスおよびそれに関連した情報の記録に，たとえば5バイトの情報が必要なので，問題を実際に1秒間走らすと，そのマシンは1000万バイトのアドレス・ストリーム・データを発生する．たった数秒間で，1巻の磁気テープが一杯になる．このことは，トレースされるべきプログラムを注意深く選択することのみならず，そのプログラムの活動度の高い部分を前もって識別することもまた重要であることを，はっきりと示している．それはまた，大量のアドレス・ストリーム・データを供給されるであろうシミュレータにおける効率の必要性を強調している．いったん，1つのアドレス・ストリームを捕捉したら，それはいろんな方法で解析されうる．たとえば，その命令およびデータのいずれが最も頻繁に参照され，そしてそれらが1つのプログラムの大きさのどの位の割合を表わしているか，を決定することが可能である．また，生起するページ・フォールトの数を最小にするためには，実行中の各時点でプログラムのどの部分が主記憶装置内に同時に存在していなければならないかを決定することが可能である．発生するページ・フォールトの数とプログラムの実行速度に関して，異なるページ置換えアルゴリズム(10-6節参照)の効果を調べることもできる．最後の例としては，プログラムの実行速度，主記憶装置の要求条件，ページ転送数などに対するページ・サイズの効果を調べるためにアドレス・ストリームを利用することなどがあげられる．次の節では，この性質の調査を行なうためのアドレス・ストリームの表現法について述べる．

アドレス・ストリーム・データ収集の性質についてのここでの短い説明を終わるにあたって言及しておくべきことは，その問題は計算機科学にとって，またあるハードウェアおよびソフトウェアの実際の設計にとって基本的ではあるが，応用プログラムあるいはシステムの利用者あるいは設計者は不経済なアド

レス・ストリームの実験を実際に行う必要はないということである．そのかわりとして，発表されている数多くの研究結果(BA1, BE1, CO1, CON, FI, GI, KE2, LE 参照)やこの本のような要約の恩恵を被むることができよう．これらは，いったん理解されプログラミングの実際に用いられると，仮想記憶システムの性能を良くすることができるプログラミングの様式の指針を与えている．

10-5　アドレス・ストリームの特性記述：局所性とワーキング・セット［BE 2, DE 1, LE, ST 参照］

1つのアドレス・ストリームは，N 個の整数値を含む1つのベクトル A として表現できる．アドレスが n ビットであるとすれば，ストリーム中の各アドレスは 2^n の可能な値をもつアルファベットの1要素である．ページ・サイズを 2^S 語とすると，アドレス空間は $P_n = 2^{n-s}$ ページを含む．先の例では，実効アドレス・サイズ(n)は24ビットの長さであった，これからアドレス空間は 2^{24}，すなわち16,777,216番地となった．ページ・サイズ(2^S)は 2^{12}，すなわち4096語であり，アドレス空間は $P_n = 2^{24-12}$，すなわち $P_n = 4096$ ページを含んでいた．語アドレスのアドレス・ストリーム A は関係

$$X = \lfloor A \div 2^S$$

によって，**ページ・アドレス・ストリーム** X に容易に変換される．すなわち，各ページ・アドレスは，語アドレスをページ・サイズで割り，最も近い等しいかあるいは小さい整数に丸めることによって得られる．たとえば，8語のページ・サイズ，すなわち $s = 3$ として，ストリーム A が与えられると，次の X のストリームが上の式から得られる．

A	36	180	38	184	40	188	44	192
X	4	22	4	23	5	23	5	24

以後の話のほとんどの部分では，アドレス・ストリームよりも(少なくともひとたびページ・サイズが選ばれたら)むしろページ・アドレス・ストリームに興味を持つことになろう．

さてここで基本的な質問に至る．前の議論から，**局所性**の直観的な観念を明らかにし，多くのプログラムは，アドレスのかなり長い系列に対してアドレス空間の小さな部分にとどまるという，強いあるいは“良好な”局所性を示すことがわかっているという経験的な事実を述べた．ここで，アドレス・ストリー

ムから得ることができる，局所性のある正確な量的尺度を求める．そうするとそれによって，たとえば，同一の問題に対する2つのプログラム——各プログラムは異なるアルゴリズムを用いている——の局所性を比較することができる．1つの考えられる尺度はページ参照の頻度分布である．高い局所性は少数のページにおける非常に高い相対頻度として現われると考えてもよいかもしれない．しかし，これは適切な尺度でないことが容易に示される．たとえば，2つのページ・アドレス・ストリーム

A	8	8	8	8	8	15	15	15	15	15
B	8	15	8	15	8	15	8	15	8	15

を考えよう．これらのストリームの両方とも，同じ頻度分布を持っているが，ストリーム A はそのほとんどの時間ただ1つのページへのアクセスだけを要求しており，一方ストリーム B はページ8と15の間の急速の交番に対して用立てるために2ページを要求している．明らかに，たとえこれら2つが同一の頻度分布を持っていても，ストリーム B の局所性はAのそれより劣っている．頻度分布のみでは局所性の適切な尺度とならないということは驚くにあたらない．というのは局所性はストリームの**微細順序**構造に依存しているからである．頻度分布はそれらが記述しているデータにおける順序を完全に無視している．

局所性の良好な記述子とは，プログラム実行の比較的長い期間にプログラムのどの部分(あるいはページ)が参照されているかを識別するものであろう．1つのそのような尺度はプログラムの**ワーキング・セット**と呼ばれている．ワーキング・セット・サイズはその期間にいくつの**相異なる**ページが参照されているかを示すものである．プログラムの実行につれて，そのワーキング・セットは変化していく．たとえば，初期化の間に参照されるプログラムの部分は主要ラインの実行の間で参照される部分とは異なるであろう．プログラムは，もしそれが小さなワーキング・セットを持っているなら，すなわちその平均ワーキング・セット・サイズが小さいならば，良好な局所性を有しているといえる．ここで，ワーキング・セットを測定する1つの技法を考えよう．

あるプログラムの実行ストリームの i 番目の参照における，すなわち，ページ・アドレス・ストリーム・ベクトル内の値 X_i における，ワーキング・セットは，集合記法

$$W(i, h) = \{X_j | j \in ((i - h) + \iota h)\}$$

と表わせる．ここに，ιh は1から h までの連続整数のベクトルである．言葉で言えば，ワーキング・セット $W(i, h)$は，位置 i におけるページからストリームを振り返って見て，長さ h ページの"窓"内に現われる異なるすべてのページの集合である．それゆえ，ワーキング・セットの大きさは選択した窓の大きさ h とストリーム内の参照点 i の両者の関数である．ワーキング・セットのサイズは $\rho W(i, h)$ と記され，ページ・アドレス・ストリームのサイズは ρX である．ワーキング・セットのサイズの期待値すなわち**平均サイズ**は，共通の窓サイズを持つすべてのワーキング・セット・サイズの和を取ることによって得られる．したがって，

$$w(h) = \frac{1}{\rho X}\sum_{i=1}^{\rho X} \rho W(i, h)$$

与えられたストリームに対する平均ワーキング・セット・サイズは，窓サイズ h の関数であることがわかる．実例の計算として**図 10-5.1** を見られたい．平均ワーキング・セット・サイズは次の関係を満たすことが示される．

1. $1 < w(h) < \min(h, 2^{n-s})$
2. $w(h) \leq w(h+1)$
3. $w(h+1) + w(h-1) < 2w(h)$　　（下方に凹）

言葉で言えば，これらの関係は，窓サイズ h に対する平均ワーキング・セットは1より小さくはなり得なく，また h あるいは 2^{n-s} のいずれか小さいもの，すなわちアドレス空間における異なるページの数より大きくはなり得ない，ということを述べている．窓サイズ $(h-1)$ および $(h+1)$ に対する平均ワーキング・セットの結合は窓サイズ h に対する平均ワーキング・セットの2倍より大きくはなり得ない．これらの性質から**図 10-5.2** に示される曲線の一般形が出て来る．

ページング・システムにおけるプログラムの局所性を表現する一般的な技法は，**図 10-5.3** に示された曲線をいくらか変形したものである．この型の曲線は，示されたような形になることがしばしば経験的に見い出されるが，ほとんどのプログラムは良好な局所性を持っており，実行に際して全プログラム・サイズより小さい主記憶装置を要求していることを示している．また，主記憶装置の2つの水準の間で，比較的低いページ転送率が要求されていることがわかる．曲線上の急な屈曲点における主記憶装置サイズはプログラムの**パラコル**

インデックス →	1	2	3	4	5	6	7	8	9	10
X →	4	22	4	22	4	23	5	23	5	24

（a）　ストリーム

	$h=2$		$h=5$		$h=10$	
i	$W(i, 2)$	$\rho W(i, 2)$	$W(i, 5)$	$\rho W(i, 5)$	$W(i, 10)$	$\rho W(i, 10)$
1	4	1	4	1	4	1
2	4,22	2	4,22	2	4,22	2
3	4,22	2	4,22	2	4,22	2
4	4,22	2	4,22	2	4,22	2
5	4,22	2	4,22	2	4,22	2
6	4,23	2	4,22,23	3	4,22,23	3
7	5,23	2	4,5, 22,23	4	4,5 22,23	4
8	5,23	2	4,5 22,23	4	4,5 22,23	4
9	5,23	2	4,5,23	3	4,5, 22,23	4
10	5,24	2	5,23,24	3	4,5,22 23,24	5

（b）　$h=2,5,10$ に対する計算表

$$w(2) \;= 19/10 = 1.9$$
$$w(5) \;= 26/10 = 2.6$$
$$w(10) = 29/10 = 2.9$$

（c）　計算された平均

図 10-5.1 平均ワーキング・セット・サイズの計算

(parachor)† と呼ばれることがあり，その効率的なワーキング・セットを反映している．主記憶装置がこの点よりあまり減らされると，ページ転送率は非常に高くなることがわかる．このことは，多数の現在活動中のページが比較的少数の主記憶装置のページ・フレームを求めて競合していることを示しており，この状態は**スラッシング**（ころげまわること）と呼ばれる．10-6 および 11-5 節でスラッシングをさらに議論しよう．

個々のどのプログラムに対しても図 10-5.3 に示された軸をもつ曲線の図を

† パラコル：　岩波理化学辞典（岩波書店）参照

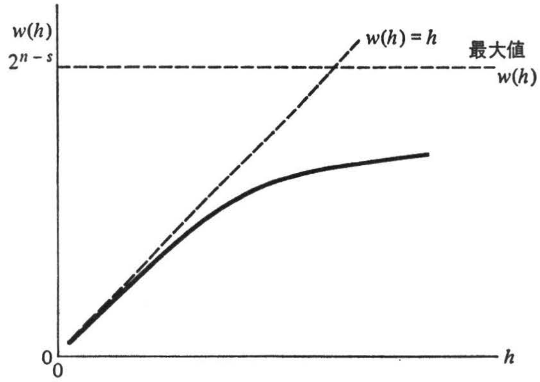
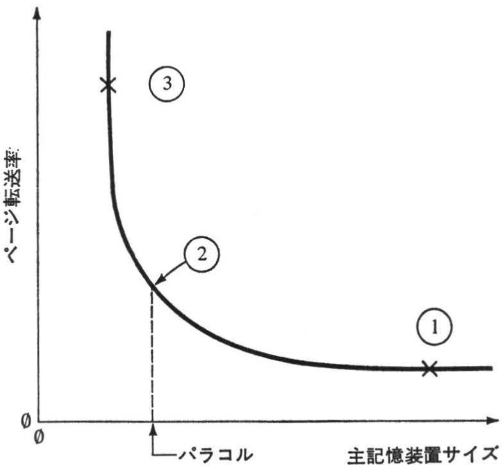


図 10-5.2　平均ワーキング・セット・サイズの曲線の一般形



1. 曲線上のこの点は，そのプログラムのために宣言されたすべての記憶装置が主記憶装置として利用できる場合の，プログラム実行を表わしている．ページ転送率は無視できる．

2. 曲線上のこの点は，あるプログラムの平均ワーキング・セット要求を指示している．宣言されたよりもかなり少ない主記憶装置しか利用できないが，ページ転送率はなお合理的である．

3. この点はスラッシング領域にあり，非常に高いページ転送率を表わしている．

図 10-5.3　参照の局所性の曲線

描くことができる．作図のためのデータを得るには，異なる主記憶装置の容量を与えその各場合においてプログラムを連続的に実行し，各実行でのページ転送率を測定する必要がある．概念は簡単であるが，実験はマシンのリソースおよび必要とされる時間で高くつく．プログラムの様式とプログラムの振舞について述べる10-7節では，そのような実験のいくつかの例を示す．

仮想記憶装置の文献では，**パラコル**および**最適記憶装置**という用語が**ワーキング・セット**の代りに用いられることがある．しかしながらワーキング・セットがもっとも一般的であると思われる．ワーキング・セットの測定は仮想記憶装置システムの正当化および初期設計のための実験的基礎を形成した．

10-6　ページ置換えアルゴリズム[AH, HE, BE 1, KI 2, MA 2 参照]

いままでに示された概念は仮想記憶装置のシステム設計環境に関連したものであった．これは文献での歴史的実際に従っている．置換えアルゴリズムの記述はCPU 設計環境の2水準階層および用語を用いてなされるであろう．その後で，またシステム設計環境に対する置換え“機構”のいくつかを述べよう．

図 10-6.1 は2水準階層と使われる用語を示している．先に述べたように，小型の高速記憶は**バッファ**と呼ばれる．そしてより大型のものは**バッキング**記憶と呼ばれ，これはアドレス空間で識別されうる．プログラムが実行されるにつれて，アドレス空間の種々の部分が活性になり，一方では以前に活性であった他の部分が眠った状態になる．良好な局所性の期間では，ほとんどの参照は

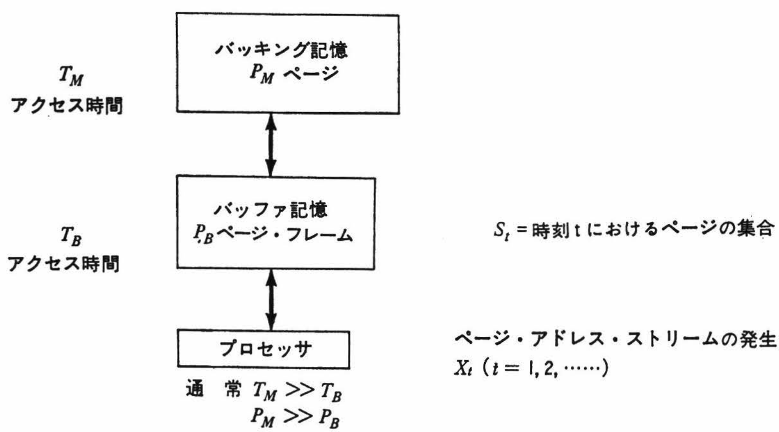


図 10-6.1　2水準階層

すでにバッファ記憶に存在しているページに対してである．これらの期間ではシステムのアクセス速度は高速のバッファ記憶のアクセス速度である．しかしページ・フォールトが起こると，プログラムが進行できるようにするためにはその前に新しいページがバッキング記憶からバッファへ移動されなければならない．バッファが一杯ならば，現在バッファ記憶に存在するページのいずれを置換えるべきかを決めるために**ページ置換えアルゴリズム**が用いられねばならない．

もちろん，参照データの“ページ・イン”が本質的な操作である．それと反対方向のページ移動にはいくらかより多様な可能性がある．その移動にかかわるページがその最後のページ・イン以来変更されていないなら，移動をまったく行なう必要がない．というのはこの場合より低速な記憶上の元の写しは今なお有効であるからである．そのページが変更されているなら，より低速な記憶上の写しは新しい内容を反映するために変更されなければならない．現在これを行なう方法には基本的に2つある．その第1は，**図 10-3.2** と関連してすでに述べられたように，ハードウェア写像のあるいは記憶装置アクセスの機構が，ページ内で変更が行なわれたことを示す変更ビットを，維持するものである．ページが置換えられることになると，そのビットがチェックされる．もしON なら，そのページは低速の記憶(ディスク)内へ写し戻される．さもなければ，この写しの操作は行なわれない．この手法はシステム環境の仮想記憶装置でよく使われている．もう1つの別の手法は，CPU 設計環境で使われているが，変更が行なわれるごとにページの変更された情報をより低速な記憶に格納することである．これは，ページ全体の“ページ・アウト”が決して必要でないことを意味している．おのおのの手法は(後に議論されるように)それが使われる環境の速度範囲で適切なものとなる．

ページ置換え操作は，普通，仮想記憶装置システムで最も不経済なオーバヘッドを伴う不利な点となっている．この理由のために，ページ置換えアルゴリズムを審査するための性能指標はそれらが要求するページ置換えの総数である．当然，アルゴリズムの比較は同一の設備と負荷でもっぱら行なわれるべきである．

次の記法を使うとする．

C = バッファ容量(ページ・フレームの数)

t = ページ・アドレス・ストリーム——ここではこれを X と呼ぶが——内の指標

$S_t(C)$ ＝ X における参照 $t(X_t)$ でのバッファ内のページの集合

$\rho S_t(C)$ ＝ $S_t(C)$ 内のページの総数

Γ_t ＝ $X_1, X_2, \cdots, X_t$ における**相異なる**ページの集合

$\gamma_t = \rho\Gamma_t$ ＝ $X_1, X_2, \cdots, X_t$ における相異なるページの総数

$A(S)$ ＝ S 上で作用する置換えアルゴリズム A によって置換えるために選ばれたページ

T_M ＝ バッキング記憶のアクセス速度

T_B ＝ バッファ記憶のアクセス速度

P_M ＝ バッキング記憶の容量(ページ)

P_B ＝ バッファの容量(ページ・フレーム)

ここでは**デマンド・ページング**のみについて触れよう．これは，すなわち，非駐在のページが参照された時点での要求のみに従ってページがバッファから持ち込まれるものである．良好な置換えアルゴリズムを用いると，実行中まったく参照されないページは決してバッファ空間を占めることはなく，まれに参照されるページは短期間の間のみバッファを占め，そして，激しく参照されるページ(したがってワーキング・セット・ページ)はバッファ内に生き残る傾向となろう．

さて，ページ・アドレス・ストリーム X を考えよう．デマンド・ページング過程は次の集合記法ステートメント(各場合に1つ)によって簡潔に表わしうる．**現在ページ**という用語は，まさに参照されているページを指す．

1. $X_t \in S_{t-1}(C)$ ならば，$S_t \leftarrow S_{t-1}$
   もし現在ページがバッファ内にあるなら，新たなバッファ集合はもとのままである．アクセス時間は T_B である．

2. $X_t \notin S_{t-1}(C)$ で $\rho S_{t-1}(C) < C$ ならば，$S_t(C) \leftarrow S_{t-1}(C) \cup \{X_t\}$
   もし現在ページがバッファ内になく，バッファが一杯でないなら，現在ページはバッファに送られる．アクセス時間は $T_B + T_M$ である．

3. $X_t \notin S_{t-1}(C)$ で $\rho S_{t-1}(C) = C$ ならば，
   $$S_t(C) \leftarrow (S_{t-1}(C) \sim A(S_{t-1}(C))) \cup \{X_t\}$$
   もし現在ページがバッファ内になく，バッファが一杯であるなら，A と名付けられたアルゴリズムが削除すべきページを決定する．それから，X_t がバッファに送り込まれる．アクセス時間は置換えられるページがバッファ内にある間に変更されたか，されなかったかに従って，$T_B + T_M$ あるいは $T_B + 2T_M$ である．

うまいページ置換えアルゴリズムというものは，出来るだけ少ないページの移動ということに帰着する．個々の置換えアルゴリズムを議論する前に，アルゴリズムの出来映えの尺度を定義することは有益である．

$P=$ バッファ内でページ要求が満たされる確率(相対頻度)，としよう．これは成功あるいは**ヒット**関数と呼ばれることがある．

$1-P=$ 望みのページがバッファ内になく，そしてそれゆえバッキング記憶にアクセスを必要とする確率．

平均アクセス時間は，バッファおよびバッキング記憶へのアクセス時間にそれらの確率を乗じ加え合わせることによって得られる．すなわち，

$$\bar{T}=PT_B+(1-P)T_M \tag{1}$$

システムのアクセス速度を，(全記憶装置がバッファ速度と同じであったとした)最大可能アクセス速度に対して相対的に表わすと，

$$S=\left(\frac{1}{\bar{T}}\right)T_B=\frac{\bar{S}}{S_B} \tag{2}$$

式(1)を $\bar{T}$ で除し，式(2)に代入し，P について解くと，

$$P=\frac{1-ST_M/T_B}{S(1-T_M/T_B)}$$

図 10-6.2 は，T_M/T_B の種々の値に対する P 対 S の図を示す．すなわち，2つの記憶の速度比が与えられたとして，曲線は望みの平均速度比 S を得るための必要成功確率を，いろいろな速度比 T_M/T_B に対して，示している．

実際の置換えアルゴリズムを研究するにあたって，“MIN” と呼ばれるものから始めよう．というのは，それがページ置換え総数に関して**最良**のアルゴリズムだからである． MIN は最良であるが，それはページ・ストリーム中で未来での参照の先見的知識を要求するので，実用的ではない．置換えが要求されると，MIN はバッファ内の各ページがページ・アドレス・ストリーム X 中で次にいつ出現するかについて調べる．MIN は，置換えるものとして，X 中で次に参照されるのが最も遠い未来となるページをバッファ内から選ぶ．記号的記法で表わすために，

$\tau_t(i)=$ バッファ内のページ i とそのページのページ・アドレス・ストリーム中での次の出現 t との(ページ数で測った)距離

$j=$ (MIN アルゴリズムによって選択されて)置換えられること

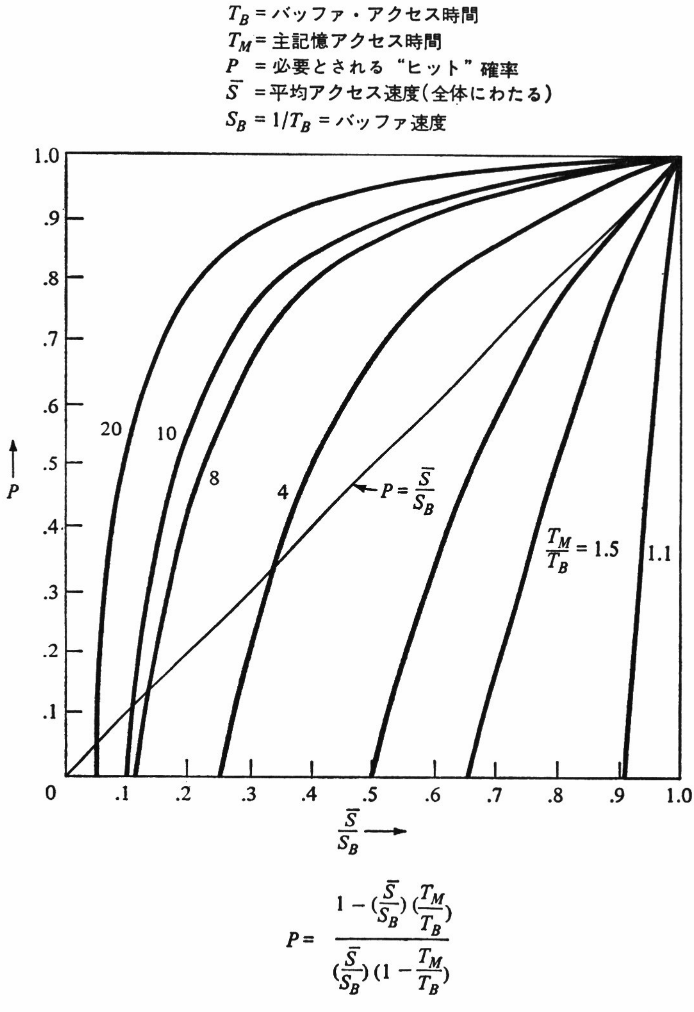


図 10-6.2　種々の速度比に対して必要とされる "ヒット" 確率

になったバッファ内のページ,

としよう．MIN は，S_t から X 中で次に参照されるのが最も遠い未来となるページを選ぶので(図 10-6.3 参照)，これは記号的に

$$j \leftarrow \underset{i}{\mathrm{Max}}\, \tau_t(i \in S_t)$$

インデックス→	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
ストリーム→	5	4	2	8	4	1	4	3	1	5	4	2	4	5	3	8	**4置換え操作**
	5	5	5	5	5	5	5	5	5	5	5	5	5	5			
バッファ	–	4	4	4	4	4	4	3*	3	3	3	2*	2	2			
	–	–	2	8*	8	1	1	1	1	1	4*	4	4	4			

(*a*)　MIN アルゴリズム

インデックス→	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
ストリーム→	5	4	2	8	4	1	4	3	1	5	4	2	4	5	3	8	**8置換え操作**
	5	5	5	8*	8	8	8	3*	3	3	4*	4	4	4	4	8*	
バッファ	–	4	4	4	4	4	4	4	4	5*	5	5	5	5	5	5	
	–	–	2	2	2	1*	1	1	1	1	1	2*	2	2	3*	3	

***そのページがバッファ内に丁度置かれた.**

(*b*)　LRU アルゴリズム

図 10-6.3　2つの置換えアルゴリズム

と表わしうる．X_j は置換えられるべき S_t 内のページである．APL 言語では，S および X をベクトルとみなすことができ，T はストリーム X 中の現在の参照に対する指標である．

$$I \leftarrow T + \lceil/(T \downarrow X)\iota S \qquad (3)$$

式(3)は S 内のページに対する最初の参照のうち，最も後のものを指す X 中の指標 I を与える．

$$\text{もし}\quad I \leq \rho X \text{ なら,}\quad J \leftarrow S\iota X[I] \qquad (4a)$$

$$\text{もし}\quad I > \rho X \text{ なら,}\quad J \leftarrow 1 \uparrow ((\rho X) < T + (T \downarrow X)\iota S)/\iota\rho S \qquad (4b)$$

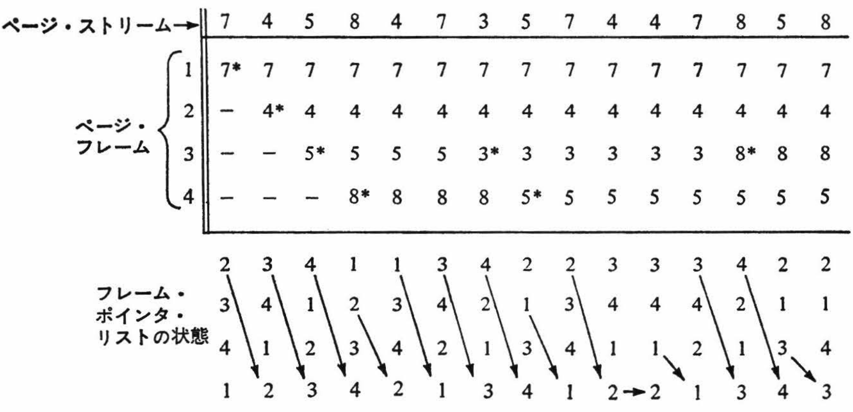
ここで第2の場合は，$X[T]$ より前方での X の内のどこにも，S のいずれの要素に対応するものがない場合に起こる．

MIN アルゴリズムは，ページ置換えの数に関しては最適であるが，ページ・アドレス・ストリームの先見的知識を必要とするので実用的では**ない**．このような知識はシミュレーションによる設計研究では利用できるが，プログラム実行時には利用できない．また MIN アルゴリズムは，変更を受けていない

ページがスワップアウトされない場合には，最適でないかもしれない．しかし，実験および設計において，MIN アルゴリズムは考慮中の置換えアルゴリズムと比較するのに有用である．図 10-6.3 は，1つの短いアドレス・ストリームと容量 P_B が3ページ・フレームのバッファに対する MIN アルゴリズムの作用を示している．

LRU(Least Recently Used)置換えアルゴリズムは実用的なものであり，MIN と概念的な関連がある．実際，距離がページ・アドレス・ストリームの未来(右)に向かってというよりむしろ左(過去)へ向かって測られるという点を除いては，それは MIN と同一である．したがって，LRU は S 内のページのうち最後の参照が最も以前のもの，すなわち，LRU のページを置換える(図 10-6.3 参照)．LRU は，過去の参照にのみ依存しているという事実によって，実用的なアルゴリズムのための1つの良き基礎となっている．LRU のインプリメンティションの機構については図 10-6.4 を見られたい．

個々のページ置換えアルゴリズムをシミュレートする際，異なるバッファ容



アルゴリズム．

ストリーム内の次のページ参照をフレーム内のすべてのページと比較する．

場合Ｉ：　不合致

フレーム内のページで，その位置がポインタ・リストの最上位で与えられるものを置換え，それからポインタ・リストを1位置上へ回転する．

場合II：　合致

合致するフレームが位置 j にあるとしよう．ポインタ・リスト内で j を見つけ，それをリストの最下位に移し，それからそれより下のすべてのポインタを1位置上へ回転する．

図 10-6.4　回転ポインタを持つ LRU アルゴリズムのインプリメント

量に対する結果を得るのが望ましいことが非常に多い．**スタック**置換えアルゴリズムは包含性という特徴を持つ1つの部類となるが，その包含性によってアドレス・ストリームの単一パスを用いてバッファ・サイズの異なる多くのシステムをシミュレーションすることができる．

$S_t(C)$を時刻tにおける容量Cのバッファ内のページの集合としよう．するとLRUは，常にそのバッファがC個のMRU(Most Recently Used)のページを含むように強いるので，

$$S_t(1) \subset S_t(2) \subset \cdots S_t(\gamma_t) = S_t(\gamma_t + 1) = \cdots \qquad (5)$$

を満足する．γ_tは$X_1, X_2, \cdots, X_t$における相異なるページの総数に等しいことを思い出そう．さらに，

$$1 \leq C \leq \gamma_t \text{ に対して} \qquad \rho S_t(C) = C$$

また，

$$C \geq \gamma_t \qquad \text{に対して} \qquad \rho S_t(C) = \gamma_t$$

である．これらの式は次のことを示している．すなわち，容量Cページのバッファに適用されたスタック・アルゴリズムは，ストリーム中の任意の点tにおいて，その容量がCページより小さいと考えたバッファの中に存在すべき集合を包含する，ページの集合を見いだすであろう．当然の結果として，どんな単一スタック・アルゴリズムも，ページ・ストリームの単一のパスを用いて，容量1, 2, , ···, Cページに対してシミュレーションされうることになる．シミュレーションにおけるこの効率は決して重要性の小さいものではない．というのはストリームは非常に大きなデータの収集物であるからである．

スタック・アルゴリズムの例にはMINとLRUが含まれる．スタックの部類に入ら**ない**アルゴリズムには，**先着順方式**(first-in-first-out)やランダム置換えがある．**図10-6.5**には，LRUが容量2および3ページ・フレームのバッファに対して，どのようなスタック・アルゴリズム包含性を示すかが示されている．

LRUアルゴリズムに近似の多くの変形が，実存する仮想記憶装置システムにおいてインプリメントされている．どの場合でも，LRUの主要な目標はページ置換え総数を最小化することである．この目標を最も簡単に実現するには，置換え選択の際，変更ページをその対象としないでおく．CPU設計環境では，このことは普通何の問題も提起しない．この場合には，**貫通記憶**，すなわちバッファ内のページを変更するのと同時にバッキング記憶内のページを更

$t\to$	1	2	3	4	5	6	7	8	9	10	11	12	13	
$X\to$	a	h	e	d	e	g	h	h	h	g	d	h	g	
$-\infty$	a 0*	a 1	a 2	d 0*	d 1	d 2	h 0*	h 0	h 0	h 1	h 2	h 0	h 1	$S_t(3)$ 集合
$-\infty$	$-\infty$	h 0*	h 1	h 2	h 3	g 0*	g 1	g 2	g 3	g 0	g 1	g 2	g 0	
$-\infty$	$-\infty$	−0	e 0*	e 1	e 0	e 1	e 2	e 3	e 4	e 5	d 0*	d 1	d 2	

(*a*) c≡3　置換え数≡7

$X\to$	a	h	e	d	e	g	h	h	h	g	d	h	g	
$-\infty$	a 0*	a 1	e 0*	e 1	e 0	e 1	h 0*	h 0	h 0	h 1	d 0*	d 1	g 0*	$S_t(2)$ 集合
$-\infty$	$-\infty$	h 0*	h 1	d 0*	d 1	g 0*	g 1	g 2	g 3	g 0	g 1	h 0*	h 1	

(*b*) c≡2　置換え数≡9

注：　1. バッファ$S_t(c)$の状態はストリーム内の各ページ参照における2重線の下の垂直欄にある．$S_t(c)$内の各要素に対して，ストリーム内のその後向き相対距離もまた示されている．置換えは＊によって示されている．
　　　2. あらゆるtに対する$S_t(2)\subset S_t(3)$によって包含性が示されている．

図 10-6.5　LRU アルゴリズム(スタック・アルゴリズム)に対する包含性の図解

新するハードウェアを設計することは理にかなっている．そのような設計をすれば，変更ページが置換えられる際に，何らページ・アウト操作を必要としない．システム設計環境においては，貫通記憶は実現できない．そして，変更ページを置換えるには，主記憶装置とバッキング補助記憶の間の非常に低速なI/O操作(ページ・アウト)を必要とする．システム設計環境置換えアルゴリズムのいくつかは，変更ページを優遇するための選択過程に重きを置き，それによってページ・アウトI/O活動を減らそうとしている．

システム設計環境におけるデマンド・ページングの特性は，CPU環境のために前に説明したものとまさに同じである．システム設計環境の用語でいえば，

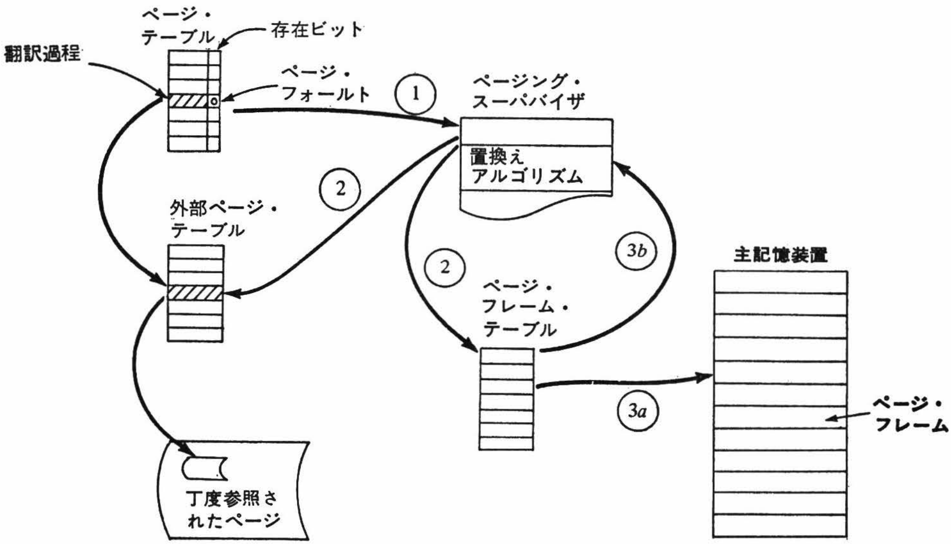
1. 主記憶装置内にすでに存在しているページ内で参照されたデータあるいは命令は(写像装置による翻訳後)ほとんど遅延なく処理される．
2. 主記憶装置内に現在存在していないページが参照され，ページ・フレームが利用可能な時には，そのページは補助記憶から主記憶へ移動され，処理が再開される．
3. 主記憶装置内に存在しないページが参照され，いかなるページ・フレームも利用できない時には，1つの駐在ページが置換えのために選ばれ，そ

の内容が最後のページ・イン以後変更されていたら補助記憶装置へ移動されなければならない．その後で，参照されたページが主記憶装置に移動でき，処理できる．

これらの機能を行なうためには，仮想記憶装置システムは，あるページが主記憶装置内に存在しているかどうかを知り，補助記憶装置内のページの位置を識別し，(もしあれば)どの主記憶装置ページ・フレームが利用可能かを識別し，ページの内容が最後のページ・イン以後変更されたかどうかを識別し，そして，I/O 操作によって主記憶装置と補助記憶装置の間でページを移動しなければならない．LRU 置換えアルゴリズムをインプリメントするには，主記憶装置内のページがどのくらい最近に参照されたかを示す機構を必要とする．これらの機能は，何らかのハードウェアの援助で仮想記憶装置オペレーティング・システム内の特別な**プログラム**によって行なわれるのが典型的である．

セグメント・ページ方式システムにおいては，主記憶装置内のページの存在は，各ページ・テーブル・エントリ内の**存在**ビット(あるいは“無効”)ビットと呼ばれるものによって示される．この考えは 10-3 節で紹介された．もし参照ページが主記憶装置に存在していないなら，アドレス翻訳の間に，存在ビットは零となり，その結果**ページ・フォールト**が生じる．ページ・フォールトは(**プログラム割込**の形式の)ハードウェア信号であり，この信号は見つからないページを主記憶装置に移動する(ページ・イン操作)特別なオペレーティング・システム・プログラムを活動させる．この特別なシステム・プログラムを**ページング・スーパバイザ**と呼ぼう．ページ・フォールトが起こると，ページング・スーパバイザは見つからなかったページを補助記憶装置内で位置ぎめし，また(もしあれば)利用可能なページ・フレームを主記憶装置内で識別しなければならない．補助記憶装置内でページの位置ぎめをするためによく使われる 1 つの技法は，しばしば**外部ページ・テーブル**と呼ばれる一組のテーブルを用いたものである．各ページ・テーブル・エントリに対して 1 つの対応する外部ページ・テーブル・エントリがあるが，これは補助記憶装置内でのページのアドレスを与える．ページ・フォールトが起こると，ページング・スーパバイザはページ・イン過程を始めるために特定の外部ページ・テーブル・エントリを参照する．外部ページ・テーブルの一つの例は**図 10-6.6** に描かれている．

主記憶装置を管理し割付けるためには，おのおののページ・フレームの状態を識別する必要がある．それは使用中なのか，利用可能なのか？この情報は**ページ・フレーム・テーブル**と呼ばれる別の特別なテーブル内で識別されうる



1．ページ・フォールトが起ると，ページング・スーパバイザが活性化される．
2．参照されたページは外部ページ・テーブルを通じて補助記憶装置内で位置決めされる．ページ・フレーム・テーブルは空ページ・フレームがあるかどうかを調べるために参照される．
3a．空ページ・フレームが利用できる．ページ・イン操作が始めうる．
3b．空ページ・フレームが利用できない．置換え操作が必要とされる．

図 10-6.6　システム設計環境におけるデマンド・ページングとページ置換え機構

が，これはページング・スーパバイザによって作られ維持されなければならない．このテーブルは主記憶装置のおのおののページ・フレームを識別するためのエントリを有している．ページ・フォールトの場合，非占有ページ・フレームがある時には，ページング・スーパバイザはページ・イン操作を開始し，ページ・フレーム・テーブルおよび特定のページ・テーブルを更新し，そして存在ビットを ON にセットする．プログラムの実行はそのページ・インが完了したとき再開される．ページ・フォールトの際，使用されていないページ・フレームがまったくない場合には，ページング・スーパバイザは置換えられるべきページを選択し，(もしこのページが最後のページ・イン以後変更されていたら)ページ・アウト操作を開始し，それから(空いているページ・フレームが利用可能なとき)現在参照されているページのためにページ・インを開始し，影響のあるページ・テーブル，外部ページ・テーブル，およびページ・フレー

ム・テーブル・エントリを更新するであろう．図10-6.6は今述べたばかりのデマンド・ページング・ページ置換え過程を表わしている．ここで，セグメント・ページ方式システムにおいてデマンド・ページングをインプリメントするための最後の2つの詳細，すなわち，(1) 変更ページの識別法，および (2) 置換えアルゴリズム，特に一般によく使われているLRUアルゴリズムのインプリメント法を示そう．

ページの内容は，ストア操作によってのみ変更できる．各ページ・フレームに関連するハードウェア機構がストア操作によって引き起こされる場合には，ページング・スーパバイザはページ・アウトが必要かどうかを決めるのに単にこの機構の状態を調べるのみでよい．多くの仮想記憶装置システムにおいては，このハードウェアの援助のことを**変更ビット**と呼んでおり，それは，フレームのメモリ内かあるいはページ・テーブル・エントリ内で，各ページ・フレームと関連している．ページの内容が変更されるごとに，それに関連した変更ビットはハードウェアによって自動的にONに（1にセット）される．変更ビットは特権モードで実行するプログラムによってのみ調べられ，オフされるであろう．したがって，ページング・スーパバイザは置換えの決定をするに際して特定の変更ビットを調べ，ページ・イン操作後その変更ビットをオフにすることができる．

ページング・スーパバイザ・プログラムの助けとなるもう1つのハードウェアの特徴は，各ページ・テーブル・エントリ（あるいは主記憶装置ページ・フレーム）内に含まれる参照ビットである．参照ビットは，記憶装置参照がそのページに行なわれるごとに，ONにされる．すべての参照ビットがONになると，それらはOFFにされる．特権プログラムのみが参照ビットを選択的にオフにできる．ページング・スーパバイザ・プログラムはLRU置換えアルゴリズムのあらい近似をインプリメントするためにそれらを用いることができる．すなわち，ページ置換え時に，参照ビットがOFFのページのみを置換えのために選択する．

10-7　プログラミングの様式とプログラムの振舞[BR 1, BR 2, CO 1, CO 2, FI 2, FR, GU, IA 1, JO 3, MC 2, NE, VE 参照]

仮想記憶装置の主要な目標は各プログラマに大きなアドレス空間を与えることにあり，この空間は高速記憶――システム設計環境では主記憶――にはめ込むように自動的にしかも動的に折りたたまれる，ということはすでに強調されてきた．プログラミングの容易さにおいて期待される改善には，参照の局所性

の原理によるかなり良好な性能が伴うべきであろう．経験によると，仮想記憶装置を念頭に置いて書かれていない代表的な現存するプログラムでさえかなり良好な局所性を持っている．また，局所性を改善できるある簡単なプログラミング技法も確認されてきている．仮想記憶装置の性能はプログラミングの様式に敏感で，実際，ページ置換えアルゴリズムよりもこの要因に対してより敏感であるという経験的な証拠がある．この理由のために，これから，プログラミングの様式がプログラムの記憶装置参照パタンにどのように影響するか，そのいくつかの一般的な影響の仕方について触れよう．

実行中のプログラムによって作り出される記憶装置アドレス・ストリームは，次の要因に依存している．

1. プログラマに与えられた問題．
2. アルゴリズムおよび使用ソース言語．
3. プログラム・トランスレータ(コンパイラ)
4. 処理されている特定のデータ．

ほとんどのプログラムは高級言語で書かれているので，言語トランスレータ・プログラム(たとえば，コンパイラ)は，記憶装置がどのように参照されるかについて，重要な役割を果たしうる．たとえば，同一の定数が1つのプログラムの異なるページで参照される場合には，それらを参照する各ページにその定数の写しをストアした方が(記憶装置で高くつくが)有利であろう．というのは，そうしないと，その定数を含んでいるページはそれらを参照している他のページによって必要とされるからである．有利さを測るいくつかのかなり簡単な尺度が仮想記憶装置環境のために設計されたコンパイラに直ちに取入られうる．一緒に使用されることが期待されるプログラムのリンキングを配慮するような，よりこった段階も，また提案されてきている．

ほとんどのプログラマは利用可能な言語とコンパイラを使用しなければならないので，自分達が行なえる性能管理の唯一の型は良好な局所性を持つように問題を編成することである．議論の目的で特別な場合を引用するために，**図10-7.1**を考慮しよう．これは同一の結果を生ずる2つのFORTRANプログラム(入力-出力を省略)を示している．ベクトルRMINは行列Aの行最小値を与える．これらのプログラムの議論において，FORTRANは行列を**列順**にストアするということを思い出すことが重要である．プログラム(a)は各行を横切って進み，現在の行最小値RMIN(I)を行列要素$A(I, J)$が小さい時のみそれで置換える．行列Aが大きいと，このプログラムの局所性はまったく

```
      DIMENSION A(100,100), RMIN(100)
      DO 1 I = 1,100
      RMIN(I) = A(I, 1)
      DO 2 J = 2,100
      IF(A(I, J) - RMIN(I))4,2,2
    4 RMIN(I) = A(I, J)
    2 CONTINUE
    1 CONTINUE
      END
```

(*a*) 行を横切って進むプログラム

```
      DIMENSION A(100,100), RMIN(100)
      DO 1 J = 1,100
    1 RMIN(J) = A(J, 1)
      DO 2 J = 2,100
      DO 3 I = 1,100
      IF(A(I, J) - RMIN(I))4,3,3
    4 RMIN(I) = A(I, J)
    3 CONTINUE
    2 CONTINUE
      END
```

(*b*) 列を下方に進むプログラム

図 10-7.1 行列の行の最小値を求める 2つのプログラム

貧弱となろう．というのは *A* の連続した要素は列下りにストアされているが，プログラムは行を横切って進んでいるからである．その時，行列 *A* の同一行内の連続した参照が異なるページになる見込が十分にある——これは貧弱な局所性である．プログラム(*b*)もまた行の最小値を求めているが，それを列下りに進むことによって行っている．プログラムは *A* の列1を RMIN に移すことから始まる．列2の要素 *I* が RMIN(*I*) と比較され，小さい方が RMIN(*I*) の値になる．列2がそのように処置された後で，プログラムは列3，などと進んでゆく．このプログラムもまた行最小値を求めているが，行列 *A* への連続的なアクセスは行列がストアされているのと同じ順序——連続する**列**要素に対してである．したがって，その局所性はプログラム(*a*)の局所性よりもずっと良好である．

いま述べた例は言語の特性，この場合には FORTRAN の行列要素のストア

の仕方，が生成されたプログラムの局所性にどのように影響しうるかを示している．この問題に対して良好な局所性を達成するためには，プログラマはFORTRANが行列をどのようにストアするかを知らなければならない．同様に，COBOLやPL/Iのような他の言語も，プログラムの局所性，したがって仮想記憶装置システムにおけるその性能に影響する特性を有している．

良好な局所性を得るために使われるいくつかのプログラミング技法には，次のものが含まれている．

1. 可能な時はいつも，それを参照する命令語のそばにデータを置く．
2. 頻繁に呼ばれるサブルーチン(たとえば，平方根ルーチン)は，コールするよりむしろプログラム中に展開する(インライン展開).
3. エラー・ルーチンのようなあまり起こらない状態のためのルーチンはプログラムの主要部分から，できれば別のモジュールとして，分離する．

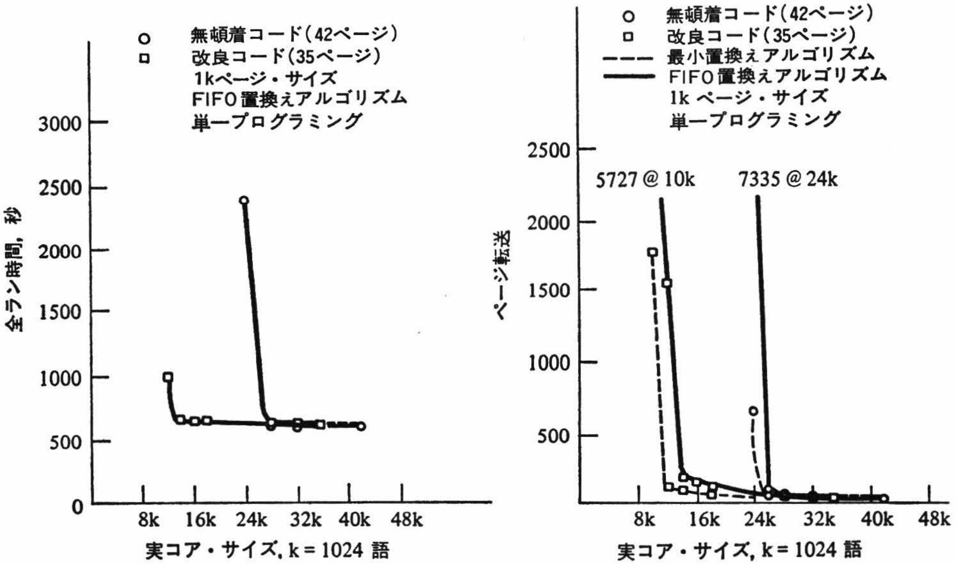
多くのプログラミング技法はアドレス空間(仮想記憶装置)を犠牲にして良好な局所性を得ている．一般に，モジュラー・プログラミングは結果として良好な局所性を生む．しかしながら，モジュールの内容は問題志向であるべきで，機能志向であるべきでない．たとえば，独立した単一のモジュールにすべてのI/O機能を配置することは，おそらく本質的に良好な局所性を生むことにならないであろう．

コーディング技法のほかに，さらにプログラムの構造もまた局所性に影響しうる．構造の一側面はすでに述べた．すなわち，初期化ルーチンやエラー・ルーチンのようなあまり起こらない状態のための手続をプログラムの主要ラインから分離する考え方である．良い構造を得るためのもう1つの方策は，互いに参照し合うモジュールを近くに置く，できれば，ASの同一ページ内に置くことである．多くのコンパイラは(IBMシステムではCSECTSと呼ばれる)オブジェクト-コード・モジュールを生成するが，これらはアドレス空間にロードされる時互いにリンクされる．もし，モジュールおよびモジュール間の活動を識別することが出来れば，局所性を改善しワーキング・セット・サイズの縮小を企てるために，頻繁に参照されるコードを時たま参照されるコードから分離して，同一ページ内のあるモジュールに納めることができるかもしれない．もちろん，これはプログラマがモジュールの活動およびモジュール間の関係についての前もっての知識を持っていることを仮定している．この情報は，通常，相当な努力なしには得ることができない．しかしながらソフトウェア・モニタを用いれば得られよう(6章参照)．コンパイラ生成によるセグメントを持

つセグメンティション・システムは，本来この型の構造化にかなったものといえよう．

プログラミング技法について上の程度話すと(そのうわつらをひっかいただけで)，1つの重要な疑問がわいてくる．プログラマは仮想記憶装置システムを使う際，特殊なプログラミング技法に対してどのくらいの努力を費やすべきであろうか？その答は簡単である．それは時と場合による！プログラムが一度使われ，そして捨てさられるか，あるいは，ほんの時たましか使われない場合には，その局所性を測定したり改善するために特別に苦心さんたんの骨折りをすることは正当化されないであろう．頻繁に使われるプログラムは，もしそれらのページング活動度が過度であるなら，かなりの注意を必要としよう．平均ワーキング・セット・サイズが約150kバイトのプログラムは，主記憶装置として2048kバイト利用できるマルチプログラミング環境では，おそらく何の悪影響も及ぼさないであろう．しかしながら，わずか384kの主記憶装置しかない同じ環境では，修正するよう注意を向けるのは正当であろう．局所性の研究から得た経験的なデータによると，ページング環境を無視して書かれたものも含めて，ほとんどのプログラムは十分なそして多くの場合良好な局所性を示していることがわかる．上に議論したように，2, 3の新たな考慮を守ることによって，プログラマは比較的少ない努力で通常良好な局所性を得ることができ，しかも大きなアドレス空間向きのプログラミングの便利さや利点のほとんどを享受することができる．

プログラミングの様式の性能に及ぼす効果のいくつかをよりよく理解するために，ここで，1968年にB. BrawnおよびF. Gustavsonによって発表された結果のいくつかを要約しよう．彼らの実験はニューヨーク州Yorktown HeightsにあるT. J. Watson Research Centerで実験用IBM M44/44Xシステム上で行なわれた．彼らはその実験のために3つの問題——逆行列，データ相関，およびソーティング——を選んだ．これらのおのおのに対して，3つのプログラミングの様式が使われた．“**無頓着な**(casual)”と呼ばれる第1のものは，ページング環境に何の注意も払わないでプログラミングされた．“**改良された**(improved)”と呼ばれる第2の様式は，ページング環境を考慮しているが，その方法はむしろ単純なものにとどまっている，“**最も改良された**(most improved)”と呼ばれる第3の様式では，ページング環境にとって出来る限り最良の構造を作り出すような企てがなされた，各プログラムはそのマシン上で単独でランされた．図10-7.2はFORTRANでコーディングされた逆行列問題に対する結果を示している．第2の様式は先に議論された行列アクセスの箇



(a) 実コア・サイズのパフォーマンスへの効果，逆行列問題(100×100).

(b) ページ置換えアルゴリズムの効果．逆行列問題(100×100).

図 10-7.2　Brawn および Gustavson の研究からのいくつかの結果

単な技法のいくつかを用いた．このプログラムは列順に行列を処理しており，そのことが(より小さなパラコルとして)本質的な改善に結びついた．無頓着なコーディングからページング環境に対して調整されたプログラムに進めることによって，データ相関やソートの問題に対しても同様な結果が得られた．図 10-7.2 の(a)の部分は，プログラミング技法での変更がより良い局所性とより小さなワーキング・セットに帰着するということを示している．図 10-7.2(b)は，プログラミング技法での変更の方が，プログラムを変えないで置換えアルゴリズムを変えて得られるもの——たとえそれが(10-6 節で述べた)最適 MIN アルゴリズムであったとしても——より，実質上良い改善をもたらすことを示している．MIN アルゴリズムは逆行列プログラムのインタプリティブな実行によって得られた．

この研究で，Brawn と Gustavson はまた注意深く制御されたマルチプログラミング環境でプログラムを走らせ，スラッシング・モニタをもつシステムと持たないシステムを比較した．仮想記憶装置システムは自動的にプログラムのアドレス空間を折りたたむのであるが，この研究(および同様の他の研究)はプ

ログラマが仮想記憶装置環境における性能に有意な効果を及ぼしうるということを示している.

10-8　CPU 設計環境：IBM S/370 モデル 155

［CO 3, GI, IB 2, IB 3, IB 4, IB 5, IB 6, IB 7, LI 2, SI 参照］

高速 CPU の設計の問題には，小型高速記憶が高速プロセッサとそれよりずっと低速な主記憶の間に必要とされる．伝統的に，高速記憶は特殊な方法，たとえば"命令語先見"で使われている．仮想記憶装置は，バッファ-主記憶の階層を管理する，別のもっと一般的な方法である．その正当化は多くのアドレス・ストリームの研究に基づく(GI, SI 参照).

モデル 165 とともにモデル 155 は，高速を達成するために CPU 環境の仮想記憶装置を用いた．モデル 158 および 168 も同様であるが，さらにそれに加えてシステム設計環境がインプリメントされた．次の議論ではモデル 155 についてのみ述べる.

図 10-8.1 に示されているように CPU 記憶装置の階層は，容量 1 メガバイトを典型とし，（4 ウェイ・インタリーブ）1 モジュールにつき 2100 ナノ秒のサイクル・タイムの主記憶装置と，容量 8k バイト，230 ナノ秒のサイクル・タイムの小型高速バッファ，とから成っていることがわかる．それに加えて，プロセッサの実効アドレスをバッファ・アドレスへ写像するために使われるイ

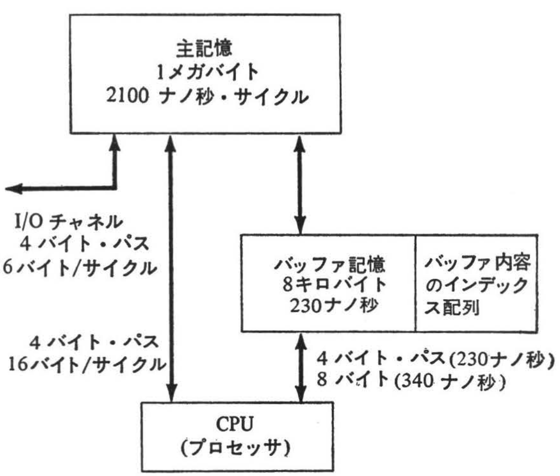


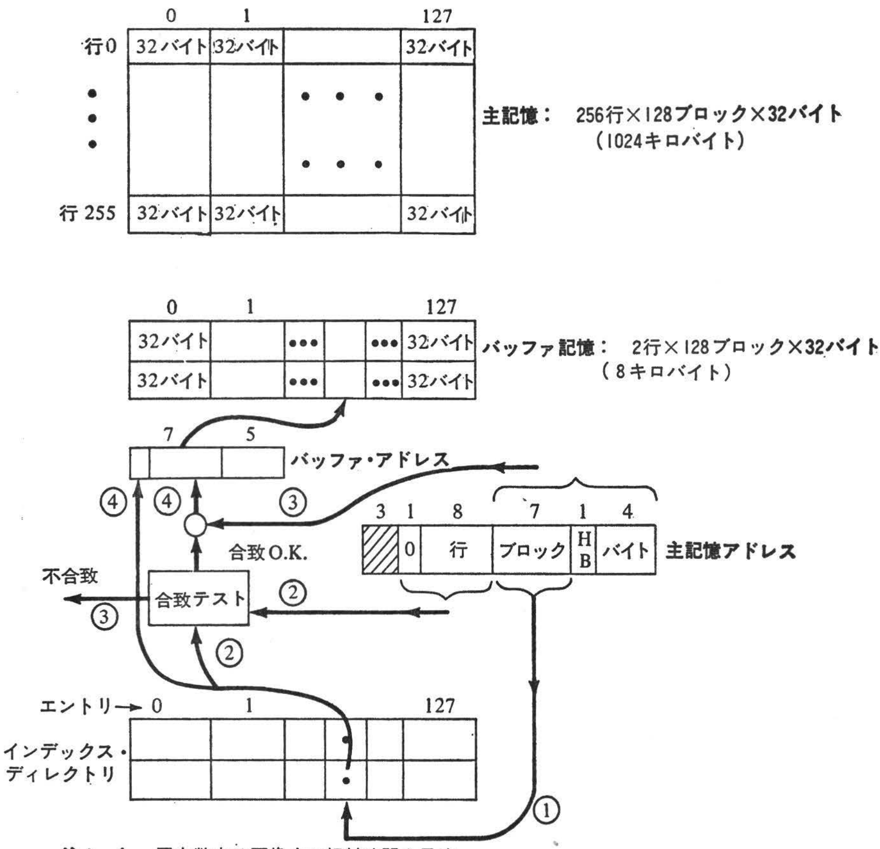
図 10-8.1　IBM システム/370 モデル 155 の編成

ンデックス・アレイを含んだ記憶装置がある.

動作中，命令あるいはデータ・アイテムが参照されるたびに，モデル155はそれがバッファ内に存在するかどうかをインデックス記憶内でチェックする．もしバッファ内にあれば，実行はバッファ速度で行なわれる．もしなければ，155が参照されたアイテムを含む16バイトの1ブロックを(主記憶装置の速度で)バッファへ移動する間の遅れが生ずる．CPUによる(フェッチに対立するものとしての)ストア参照は，バッファと主記憶の**両者**に対して行なわれる(**慣通記憶**)．これによりバッファの内容の現在の写しは常に主記憶に存在することになり，したがって，置換えが必要な時に変更情報をバッファから主記憶装置へもどす必要がまったくなくなる．この“慣通記憶”手順は実効参照速度を低下させる．この低下は統計的にはまったく小さなものになりそうである．というのは，一部には，ストア参照は比較的まれである(全体の20%が典型的である)からであり，それにまた，**書き込みアクセス**が完了する前に主記憶装置の同一モジュールを参照しようとしない限り，ストア操作は全く処理の遅れを引き起こさないからである．

モデル155におけるI/Oチャネルの参照は，典型的には次のように処理される．入力データのアドレスがバッファをチェックするために使われ，もし存在すれば，そのバッファの存在ビットはOFFにセットされ，情報は主記憶装置へ入れられる．その後でそのようなデータに対する次のCPU参照は主記憶装置からバッファへの最新の写しを持って来るということになろう．出力データ参照は主記憶装置からのみ出してくる必要がある．というのは，慣通記憶の原理は最新の写しは主記憶内にあるということを保証しているからである．

主記憶装置とバッファ記憶の間の情報の転送のために，モデル155はハードウェアによって完全に制御される一種のページング機構を使用している．バッファ記憶・主記憶の構成は**図10-8.2**に示されている．モデル155の用語では，ページはストーレッジ・ブロックと呼ばれる．各ストーレッジ・ブロックの大きさは32バイトである．主記憶装置は行編成となっており，各行は128ブロック含んでいる．1メガバイトの主記憶装置は図10-8.2に示すように256行含む．バッファ記憶もまた1行128ブロックの行編成である．8kバイトの大きさなので，バッファは2行を持つ．多対1写像機構が主記憶とバッファ記憶の間で使われる．したがって，主記憶の列0のブロックはバッファ記憶の列0にのみ置かれうる．主記憶の列1がバッファ記憶の列1にのみ置かれうる．以下同様．図10-8.2の1メガバイト記憶装置に対して，この多対1機構は256ブロックを2ブロック上に写像する．バッファ置換え戦略は，そのかかわる列に



注： 1. 円内数字は写像中の相対時間を示す．
2. 合致テストは，バッファ記憶装置中にデータの存在を示すインデックス内の，二つの正当性ビットのチェックを含む．不合致の出口は，バッファ内の1ブロックを主記憶内の要求されたブロックに置換えるための論理へと導びかれる．
3. 主記憶の列K内のブロックはバッファの列K内に存在しなければならないので，主記憶のいずれの1列のブロックも同時には高々二つしかバッファに存在できない．

図 10-8.2 モデル155における主記憶アドレスからバッファ・アドレスへの写像

対して2ブロックのみどうするかを決定するだけでよいので，LRU アルゴリズムをインプリメントするのはむしろ容易である．バッファ内に存在しない参照ブロック(ページ・フォールト)はハードウェアによるブロック置換え活動を要求する．バッファ・ブロックから主記憶へのブロック転送(ページ・アウト)の要求は慣通記憶性のために決して起こらない．

モデル155の動作中，発生されるすべての記憶装置アドレスは主記憶装置を指している．1メガバイトのモデル155では，プロセッサは各参照に対して有効20ビットのアドレスを発生する．そしてこれは8kバイトのバッファに対する13ビットのアドレスに翻訳（写像）されなければならない．翻訳のいくらかの詳細は図10-8.2に示されている．システム/360/370のマシンで例のごとく，発生される実効アドレスは，いくつかのフィールドから成っていると考えられる．ブロック(列)フィールドは写像のために使われるバッファ・インデックス記憶を指している．それはバッファ・ブロックと同じ数（256）のエントリを有している．各インデックス・エントリは主記憶装置の行を識別するが，その行のブロックの内容はバッファ内に現在ストアされている．翻訳の間，実効アドレスのブロック・フィールドはインデックスをアドレスするために使われる．それから(その列に対する) 2つのインデックス・エントリは実効アドレスの行フィールド(および存在ビット)との合致がチェックされる．合致が得られれば，(そのインデックスからの)列番号が，参照されたバイト（単数あるいは複数)を正確に位置づけする“半ブロック(half block)”およびバイト・アドレスとともに，バッファをアクセスするために使われる．図10-8.2には示されていないが，各バッファ・ブロックに対する最近の参照を示すアクティビティ・ビットは各参照ごとに更新される．これらは，置換えの必要なときに，(2つのうちの)どのブロックを置換えるべきかを決めるために後で使われる．置換えの際には，バッファ内のブロックはそのまま残しておかれるが，参照された情報を含んでいる半ブロック(16バイト)のみはただちに移動される．

バッファ-バッキング記憶管理の性能は製造会社によって示されており，実際のプログラムによる多くの実験で主記憶装置参照の90％以上がバッファ内に見い出されたという．

モデル155は，IBMの他のCPU設計環境モデルの代表である．写像機構，システム速度，および容量には多くの変形があるが，インプリメンティションの原理はごく類似している．

モデル158およびモデル168のCPU設計環境は他のシステム/360/370モデルには見られない特別な付加ハードウェアによってインプリメントされている．さらに，モデル158およびモデル168は仮想記憶装置のシステム設計環境をインプリメントするために別のハードウェアを有している．これは11章に述べられる．

演 習 問 題

1. 折りたたみの概念を論じ，非仮想記憶装置システムにおける次の例をあげよ．
   (a) プログラムの折りたたみ．
   (b) データの折りたたみ．

2. 仮想記憶装置の2つの環境は何か．そしてそれらはどのように違うか．

3. 2^p 語のページ・サイズを持つ 2^n 語のアドレス空間に対する次の写像方法を対比せよ．おのおのの場合において，必要とされる写像セルの総数を与えよ．
   (a) 連想写像
   (b) アドレッシングによる写像
   また，各方法の他の利点および不利な点について論評せよ．

4. 主記憶装置内のページ・テーブルと小さな連想記憶を用いる写像機構を論ぜよ．

5. ページ・フォールトとは何か，それはどのようにして検出され，そしてそれについて何がなされるか．

6. ページ方式のみの仮想記憶装置とセグメント方式のみの仮想記憶装置を対比せよ．

7. Burroughs 5500 の PRT にはどんな情報が含まれているか，そしてそれはどのように用いられるか．

8. 次の用語およびそのおのおのが起こるシステムの型を論ぜよ．
   (a) 内部フラグメンティション
   (b) 外部フラグメンティション

9. 写像機構に含まれることが多い次の制御ビットのおのおのの目的を述べよ．
   (a) 存在ビット
   (b) 変更ビット
   (c) アクティビティビット

10. 次のもののハードウェアによるあるいはソフトウェアによるインプリメ

ンティションの妥当性について論ぜよ．

(a)　写　像

(b)　ページ・フォールトの検出

(c)　ページ・フォールトの応答

(d)　入力/出力制御

11. セグメンティションとページングの両者を用いた典型的なシステムにおけるアドレス翻訳の過程を論ぜよ．

12. アドレス・ストリーム・トレースの各記録において捕えられるべき情報の少なくとも2つの項目を示唆せよ．

13. 出来るだけ正確に次のものを論ぜよ．

(a)　参照の局所性

(b)　ワーキング・セット

(c)　平均ワーキングセット・サイズ

(d)　パラコル

14. (a)　ページ置換えは仮想記憶装置システムにおいて何故そんなに重要な操作なのか．

(b)　置換えアルゴリズムの性能のもっとも重要な単一の尺度は何か．これは何に依存するか．

15. MIN および LRU 置換えアルゴリズムを簡潔に述べよ．それらはどのように関連しているか．

16. スタック・アルゴリズムは何か．それは何故アルゴリズムの重要な部類なのか．

17. 仮想記憶装置の性能を改善するために用いることができるプログラミング様式の少なくとも2つの技法を簡潔に述べよ．

18. 引用された実験からみて，置換えアルゴリズムとプログラミング様式のどちらがもっとも性能に影響するように思われるか．

19. (a)　仮想記憶装置のどんな環境が IBM モデル 155 にインプリメントされたか．

(b)　モデル 155 における写像方法を述べよ．

(c)　“貫通記憶”の原理は何か．それはどこで適切か．

(d)　モデル 155 において I/O 転送とバッファおよび主記憶へ(から)

のデータの移動の間の論理関係を述べよ.

20. 100ナノ秒サイクルのバッファと1000ナノ秒サイクルの主記憶をもつCPU環境のシステムを仮定せよ. 平均実効速度がバッファ速度の80%でなければならないとすると, バッファ“ヒット”の確率はいくらでなければならないか.

21. この大研究課題はアドレス・ストリームを捕捉することを必要とする. これは現存のマシンあるいは言語インタプリタを変更することによって行なうことができる. これを行なったと仮定して, 逆行列, (テキスト系列内の語数計算のような)テキスト処理, バブル(bubble)あるいはマージ・ソート, 等のようないくつかの“標準”課題をプログラムし, そのおのおのに対してアドレス・ストリームを捕えよ. それから少なくとも次のものを含むストリーム分析プログラムを書け.

(a) (たとえば)8, 16, 32あるいは64語のページ・サイズに対して, 各ページに対する読み, 書き, および全参照の相対頻度. 4つのヒストグラムを描け.

(b) 上のページ・サイズのおのおのに対してページ置換えの総数を決めるためのシミュレータを書け. LRU置換えアルゴリズム(これはスタック・アルゴリズムであることを思い出せ)と, プログラムとデータの和の最大サイズの1/8, 1/4, 1/2および1/1の高速記憶サイズを仮定する.

(c) 調査結果を要約する報告書を書け.

22. 仮想記憶装置システムが利用できるなら, 自分自身で選んだプログラムおよびデータでBrawm-Gustavson型の実験を再現せよ. 次の事柄に注意せよ.

(a) デバッギングの後, テスト・プログラムの時間測定はシステム上でこれらのプログラムのみが走っている状態で行なわれなければならない.

(b) 単一プログラム・テストに対して, プログラムとデータの和が意味のあるページングを引き起こすのに十分なほど大きいことを確かめよ.

(c) 通常, 主記憶装置の利用可能サイズを変えることはできない. しかし, 同一課題に対するプログラミング様式の効果の比較を行なうことはなお可能である.

23. S/360 モデル 85, S/370 モデル 155, 165 の種々の CPU 環境のインプリメンティションを論ずるために適切な IBM のマニュアルと文献を調べよ．類似性と差異を強調せよ．

参 考 文 献

1. [AH] Aho, A.V., P. J. Denning, and J. D. Ullman: Principles of Optimal Page Replacement, *Journal of the ACM 18*, no. 1, pp. 80-93, January, 1971.
2. [AR] Arden, B. W., B. A. Galler, T. C. O'Brien, and F. H. Westervelt: Program and Addressing Structure in a Time-Sharing Environment, *Journal of the ACM 13*, no. 1, pp. 1-16, January, 1966.
3. [BA1] Batson, A., S. M. Ju, and D. C. Wood: Measurements of Segment Size, *Second ACM Symposium on Operating System Principles*, Princeton University, pp. 25-29, October 20-22, 1969.
4. [BA2] Baylis, M.J.H., D. G. Fletcher, and D. J. Howarth: Paging Studies Made on the I.C.T. ATLAS Computer, *IFIP Proc. of the 1968 Congress 2*, pp. 831-837, 1968.
5. [BE1] Belady, L. A.: A Study of Replacement Algorithms for a Virtual-Storage Computer, *IBM Systems Journal 5*, no. 2, pp. 78-101, 1966.
6. [BE2] Belady, L. A., R. A. Nelson, and G. S. Shedler: An Anomaly in Space-Time Characteristics of Certain Programs Running in a Paging Machine, *Comm. of the ACM 12*, no. 6, pp. 349-353, June, 1969.
7. [BE3] Belady, L. A. and C. J. Kuehner: Dynamic Space-Sharing in Computer Systems, *Comm. of the ACM 12*, no. 5, pp. 282-288, May, 1969.
8. [BO] Bovet, D. P. and G. Estrin: A Dynamic Memory Allocation Algorithm, *IEEE Transactions on Computers C-19*, No. 5, pp. 403-411, May, 1970
9. [BR1] Brawn, B. S. and F. G. Gustavson: Program Behavior in a Paging Environment, *AFIPS Conference Proc., Fall Joint Computer Conference 33*, pp. 1019-1032, 1968.
10. [BR2] Brawn, B. S., F. G. Gustavson, and E. S. Mankin: Sorting in a Paging Environment, *Comm. of the ACM 13*, No. 8, pp. 483-494, August, 1970.
11. [BU] Burroughs Corp.: The Descriptor–A Definition of the B5000 Information-Processing System, *Burroughs Corp.*, 1961.
12. [CO1] Coffman, E. G. and L. C. Varian: Further Experimental Data on the Behavior of Programs in a Paging Environment, *Comm. of the ACM 11*, No. 7, pp. 471-474, July, 1968.
13. [CO2] Coffman, E. and P. Denning: "Operating System Theory," Prentice-Hall, Englewood Cliffs, N.J., 1974.
14. [COM] Comeau, L. W.: A Study of the Effect of User Program Optimization in a Paging System, *ACM Symposium on Operating System Principles*, Gatlinburg, Tenn., October 1-4, 1967.
15. [CON] Conti, C. J.: Concepts for Buffer Storage, *IEEE Computer Group News*, vol. 2, No. 8, 1969.
16. [DE1] Denning, P. J.: The Working Set Model for Program Behavior, *Comm. of the ACM 11*, No. 5, pp. 323-333, May, 1968.
17. [DE2] Denning, P. J.: Virtual Memory, *Computing Surveys 2*, No. 3, pp. 153-189, September, 1970.
18. [DE3] Denning, P. J.: On the Management of Multilevel Memories, *Proc. of the 3rd Princeton Conference on Information Science and Systems*, Princeton University, pp. 162-165, 1969.
19. [DE4] Dennis, J. B.: Segmentation and the Design of Multiprogrammed Computer Systems, *Journal of the ACM 12*, No. 4, pp. 589-602, October, 1965.

20. [DO] Doherty, W. J.: Scheduling TSS/360 for Responsiveness, *Proc. AFIPS, FJCC*, 1970.
21. [FE] Fenichal, R. R. and J. C. Yochelson: A LISP Garbage-Collector for Virtual-Memory Computer Systems, *Comm. of the ACM 12*, No. 11, pp. 611-612, November, 1969.
22. [FI1] Fikes, R. E., H. C. Lauer, and A. L. Vareha, Jr.: Steps Toward a General-Purpose Time-Sharing System Using Large Capacity Core Storage and TSS/360, *Proc. of the 23rd National Conference of the ACM P-68*, pp. 7-18, 1968.
23. [FI2] Fine, G.H., C. W. Jackson, and P. V. McIsaac: Dynamic Program Behavior Under Paging, *Proc. of the 21st National Conference of the ACM P-66*, pp. 223-228, 1966.
24. [FO] Fotheringham, J.: Dynamic Storage Allocation in the Atlas Computer, Including an Automatic Use of a Backing Store, *Comm. of the ACM 4*, No. 10, pp. 435-436, October, 1961.
25. [FR] Freibergs, I. F.: The Dynamic Behavior of Programs, *AFIPS Conference Proc., Fall Joint Computer Conference 33*, pp. 1163-1167, 1968.
26. [GI] Gibson, D.H.: Considerations in Block-oriented Systems Design, *AFIPS Conference Proc. SJCC*, pp. 75-80, 1967.
27. [GR] Greenfield, M. N.: FACT Segmentation, *AFIPS Conference Proc., Spring Joint Computer Conference 21*, pp. 307-315, 1962.
28. [GU] Guertin, R. L.: Programming in a Paging Environment, *Datamation 18*, No. 2, pp. 48-55, February, 1972.
29. [HA1] Hatfield, D. J. and J. Gerald: Program Restructuring for Virtual Memory, *IBM Systems Journal*, Vol. 10, No. 3, 1971.
30. [HE] Hellerman, H.: Complementary Replacement, a Meta Scheduling Principle, *Proc. ACM Conference on Operating System Principles*, Princeton, N.J., 1969.
31. [IB1] IBM Corp.: "Introduction to Virtual Storage in System/370," No. GR20-4260, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
32. [IB2] IBM Corp.: "IBM System/360 Model 85 Functional Characteristics," No. GA22-6916, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
33. [IB3] IBM Corp.: "IBM System/370 Model 155 Functional Characteristics," No. GA22-6942, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
34. [IB4] IBM Corp.: "A Guide to the IBM System/370 Model 155," No. GC20-1729, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
35. [IB5] IBM Corp.: "A Guide to the IBM System/370 Model 158," No. GC20-1754, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
36. [IB6] IBM Corp.: "IBM System/370 Model 165 Functional Characteristics," No. GA22-6935, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
37. [IB7] IBM Corp.: "System/370 Model 168 Functional Characteristics," No. GA22-7010, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
38. [JO1] Jodeit, J. G.: Storage Organization in Programming Systems, *Comm. of the ACM 11*, No. 11, pp. 741-746, November, 1968.
39. [JO2] Jones, R. M.: Factors Affecting the Efficiency of a Virtual Memory, *IEEE Transactions on Computers C-18*, No. 11, pp. 1004-1008, November, 1969.
40. [JO3] Joseph, M.: An Analysis of Paging and Program Behavior, *Computer Journal 13*, No. 1, pp. 48-54, February, 1970.
41. [KA] Katzan, H.: Storage Hierarchy Systems, *AFIPS Conference Proc., Spring Joint Computer Conference 38*, pp. 325-336, 1971.
42. [KE1] Keefe, D. D.: Hierarchical Control Programs for Systems Evaluation, *IBM Systems Journal 7*, No. 1, pp. 15-21, 1968.
43. [KE2] Kernighan, B. W.: Optimal Segmentation Points for Programs, *Second ACM Symposium on Operating System Principles*, Princeton University, pp. 47-52, October 20-22, 1969.

44. [KI1] Kilburn, T., D.B.G. Edwards, M.J. Lanigan, and F. H. Sumner: One-level Storage System, *IRE Transactions EC-11*, No. 2, pp. 223-235, April, 1962.
45. [KI2] King, W. F.: Analysis of Paging Algorithms, *IBM Thomas J. Watson Research Center Report RC-3288*, Yorktown Heights, New York, March 17, 1971.
46. [KN] Knuth, D. E.: "The Art of Computer Programming," Vol. 1, Fundamental Algorithms, Addison-Wesley, Reading, Mass., 1968.
47. [KU1] Kuck, D. J. and D. H. Lawrie: The Use and Performance of Memory Hierarchies: A Survey, *University of Illinois at Champaign-Urbana, Dept. of Computer Science Report No. 363*, December 4, 1969.
48. [KU2] Kuehner, C. J. and B. Randell: Demand Paging in Perspective, *AFIPS Conference Proc., Fall Joint Computer Conference,* vol. 33, pp. 1011-1018, 1968.
49. [LE] Lewis, P.A.W. and P. C. Yue: Statistical Analysis of Program Reference Patterns in a Paging Environment, *IEEE Computer Society Conference*, Boston, Mass. pp. 133-134, September 22-24, 1971.
50. [LI1] Lindquist, A. B., R. R. Seeber, and L. W. Comeau: A Time-Sharing System Using an Associative Memory, *Proc. of the IEEE 54*, No. 12, pp. 1774-1779, December, 1966.
51. [LI2] Liptay, J. S.: Structural Aspects of the System/360 Model 85, Part II, The Cache, *IBM Systems Journal*, vol. 7, No. 1, pp. 15-21, 1968.
52. [MA1] MacKenzie, F. B.: Automated Secondary Storage Management, *Datamation* 11, No. 11, pp. 24-28, November, 1965.
53. [MA2] Mattson, R. L., J. Gecsei, D. R. Slutz, and I. L. Traiger: Evaluation Techniques for Storage Hierarchies, *IBM Systems Journal*, Vol. 9, No. 2, pp. 78-117, 1970.
54. [MC1] McGee, W. C.: On Dynamic Program Relocation, *IBM Systems Journal*, vol. 4, No. 3, pp. 184-199, 1965.
55. [MC2] McKellar, A C. and E. G. Coffman, Jr.: Organizing Matrices and Matrix Operations for Paged Memory Systems, *Comm. of the ACM*, vol. 12, No. 3, pp. 153-165, March, 1969.
56. [MO] Morganstein, S. J., S. Winograd, and R. Herman: SIM/61: A Simulation Measurement Tool for a Time-Shared, Demand Paging Operating System, *ACM SIGOPS Workshop on System Performance Evaluation*, Harvard University, pp. 142-172, April 5-7, 1971.
57. [NE] Nelson, R. A. and G. S. Shedler: An Anomaly in the Space-Time Characteristics of Certain Programs Running in Paging Machines, *Comm. ACM*, vol. 12, No. 6, pp. 349-353, June, 1969.
58. [ON] O'Neill, R. W.: Experience Using a Time-Shared Multiprogramming System with Dynamic Address Relocation Hardware, *AFIPS Conference Proc., Spring Joint Computer Conference*, vol. 30, pp. 611-621, 1967.
59. [RA1] Randell, B. and C. J. Kuehner: Dynamic Storage Allocation Systems, *Comm. of the ACM*, vol. 11, No. 5, pp. 297-306, May, 1968.
60. [RA2] Randell, B: A Note on Storage Fragmentation and Program Segmentation, *Comm. of the ACM 12*, No. 7, pp. 365-372, June, 1969.
61. [SA] Sayre, D.: Is Automatic "Folding" of Programs Efficient Enough to Displace Manual? *Comm. of the ACM 12*, No. 12, pp. 656-660, December, 1969.
62. [SH1] Shedler, G. S. and S. C. Yang: Simulation of a Model of Paging System Performance, *IBM Systems Journal*, vol. 10, No. 2, pp. 113-128, 1971.
63. [SH2] Shemer, J. E. and S. C. Gupta: On the Design of Bayesian Storage Allocation Algorithms for Paging and Segmentation, *IEEE Transactions on Computers C-18*, No. 7, pp. 644-651, July, 1969.
64. [SI] Sisson, S. S. and M. J. Flynn: Addressing Patterns and Memory-Handling Algorithms, *AFIPS Conference Proc., Fall Joint Computer Conference 33*, pp. 957-967, 1968.
65. [TH] Thorington, J. M. and J. D. Irwin: A New Philosophy in Dynamic Memory Allocation, *Proc. of the 4th Hawaii Conference on Systems Science*, University of Hawaii, pp. 431-434, January 12-14, 1971.

66. [VA] Vareha, A. L., R. M. Rutledge, and M. M. Gold: Strategies for Structuring Two-Level Memories in a Paging Environment, *Second ACM Symposium on Operating System Principles*, Princeton University, pp. 54-59, October 20-22, 1969.
67. [VE] Verhoef, E. W.: Automatic Program Segmentation Based on Boolean Connectivity, *AFIPS Conference Proc., Spring Joint Computer Conference 38*, pp. 491-496, 1971.
68. [WA] Wallace, V. L. and D. L. Mason: Degree of Multiprogramming in Page-on-Demand Systems, *Comm. of the ACM 12*, No. 6, pp. 305-308, June, 1969.
69. [WE1] Weil, J. W.: A Heuristic for Page Turning in a Multiprogrammed Computer, *Comm. of the ACM*, vol. 5, No. 9, pp. 480-481, September, 1962.
70. [WE2] Weizer, N. and G. Oppenheimer: Virtual Memory Management in a Paging Environment, *AFIPS Conference Proc., Spring Joint Computer Conference 34*, pp. 249-256, 1969.

第 11 章

仮想記憶装置：システム設計環境

その要素のアクセスタイムが 1000：1(あるいはそれ以上) の比をもつメモリ階層で構成されている仮想記憶装置は，システム設計環境を特徴づける．典型的には，高速側の要素は主記憶装置であり，低速側はディスクあるいはドラムのような装置である．10 章で議論したように，スーパバイザ機能は主としてシステム・ソフトウェアによってまかなわれているが，ページの写像や参照・変更活動の記録のような非常に頻度の高い操作に対しては特別なハードウェアの助けを借りている．

さて，いくつかのアーキテクチャの選択に取りかかろう．これらおのおのはそれ自身の柔軟性とコスト-性能特性を持っている．1 つの基本的な論点は，システムが利用可能な大きなアドレス空間を現在活きているユーザ・タスクによって共有される単一の大きなユニットとして供給しているのか，あるいは**おのおの**のユーザに大きな利用可能なアドレス空間が与えられているのか，である．第一のものを選択すると，これは特に小型システム上でインプリメントするのがより容易であり，また，問題プログラムの完全な両立性が現行の通常のシステムとの間で容易に維持されうる．これらの特徴は，なにゆえ IBM システム/370 に対する最初の仮想記憶装置オペレーティング・システム(OS/VS と呼ばれる)がこの機能を使用しているかを説明するのに役立つはずである．

複雑性と互換性のスペクトルの反対の端に位置するのが Multics である．これはタイムシェアリング環境における最大限の一般性と共有を求めて無から設計されたものである．Multics はユーザのおのおのに対してユーザ自身の非常に大きなアドレス空間を供給し，プログラムとデータの共有に対しては，エレガントな記憶装置安全性機構を含めて，広範囲な備えをしている．

もう 1 つの主要なアーキテクチャの選択は，IBM VM/370 オペレーティング・システムによって代表されるものである．これは，各ユーザに仮想記憶装

置のみならず，**仮想マシン**(VM)すなわち物理的な S/370 計算機構成と機能的に正確に等価なものを供給する．各 VM は仮想 I/O その他の装置のみならず，特権命令を含んだすべての CPU 命令を有している．VM は，単に複数のプログラムをマルチプログラミングしているというよりむしろ複数の**オペレーティング・システム**全体をマルチプログラムしている．VM システムはホスト・システムのリソースの共有によって多くのオペレーティング・システムを同時に世話しているが，このことはそのオペレーティング・システムのいずれも再プログラミングすることなく行なわれる．

この章では，IBM VS, VM および GE-MIT Multics を時宜を得た例として用いながら，現存する仮想システムの主要なクラスをいくらか詳細に調べよう．

11-1 システム設計の考察 [GE, GL, HA 1, IB 2, JO, LE 参照]

セグメント・ページ方式アドレス空間を仮定しよう．すなわち，アドレス空間は最大サイズが同一のセグメントに分割され，各セグメントはページに再分割される．10-3 節に議論したように，セグメントは，写像テーブルに対して(ページのみのシステムよりも)少ない空間的なオーバヘッドで，数多くのユーザ間で情報を共有させることを可能にしている．また可変サイズのセグメントはアドレス空間割当のための便利で柔軟な単位として役に立つ．ページング機構は主記憶装置管理を単純で柔軟性のあるものとする．というのは，その場合，主記憶装置は，比較的簡単な置換えアルゴリズム(比較的とはセグメント・非ページ方式主記憶装置の管理の複雑さに比較してということ)によって利用できるページ・フレームのプールとみなすことができるからである．

前に述べたように，基本的にアーキテクチャで考慮すべきことは，通常のマルチプログラム方式オペレーティング・システムにおいて物理的主記憶装置が共有されるのと全く同様な方法で，すべてのユーザによって共有される単一の大きなアドレス空間をシステムが保持するのか，あるいはユーザのおのおのにそれ自身の大きなアドレス空間が与えられるべきであるのか，どうかである．これらの選択のうち第1のものは AS(1)型システムと呼び，第2のものは AS(n)システムと呼ぶことにしよう．AS(1) においては，スーパバイザ，常駐システム機能，およびスケジュールされるすべての実行ジョブは，同一の大きなアドレス空間を共有する．AS(1) 手法を採ろうとする主要な動機は，すでに広く行きわたって使われている“通常の”オペレーティング・システムとの互換性である．その主記憶装置が今は単一のシステム・アドレス空間に対応している．

AS(n) システムは実行中のユーザのおのおのにユーザ自身の AS を供給する．ここでの“ユーザ”とは，たとえば，バッチ・ジョブとかタイムシェアリング・タスクとか遠隔処理応用のことである．

AS(1) における写像には（アドレス空間を写像するために）1つのセグメント・テーブルを必要とし，割当てられたすべてのセグメントを写像するために，各セグメントは1つのページ・テーブルを持つ．これらのテーブルはほとんどの場合がそうであるように主記憶装置に駐在すると仮定する．セグメント・テーブル・オリジン・レジスタ(STOR)と時に呼ばれる特殊目的マシン・レジスタは，セグメント・テーブルの主記憶装置でのオリジンを識別する．セグメント・テーブルの各エントリはセグメントのページ・テーブルの主記憶装置でのオリジンを識別する．ページ・テーブルのエントリは主記憶装置内に駐在している各ページに対するページ・フレームの位置を識別する（図 10-3.2 参照）．

AS(n) システムにおいては，各ユーザは自分自身のアドレス空間を持っており，それ自身のセグメントおよびページ・テーブルによって写像される．プログラムを活性化するために，スーパバイザはマシンの STOR レジスタにそのプログラムのセグメント・テーブルのアドレスをロードする．理論的には AS(n) システムにおいて創成できるアドレス空間の数はほとんど無制限である．図 11-1.1 に概略が描かれている写像機構はこの可能性を示している†．実際問題としては，共存が許されるアドレス空間（すなわちユーザ）の数は利用可能なリソースによって制限される．たとえば，写像テーブルは主記憶装置の一部を使う．新しい各 AS 毎に，これは増加する．デマンド・ページングに対して利用可能な主記憶装置および限られた補助記憶装置のサイズと速度はまたある水準の活動度しか維持できない．これはまたユーザの数を制限するが，しかしこれは単純にはできない．というのはワーキング・セットはプログラムによって非常に異なるからである．新しい各 AS に対していくらかのディスク空間もまた必要とされる．

1つのアドレス空間のための基本的なパラメタには，(1) アドレス空間サイズ，(2) セグメント・サイズ，および (3) ページ・サイズが含まれる．多くの考慮すべき事柄がこれらのパラメタに対する値の選択に影響を及ぼす．これらの考慮すべき事柄には，オペレーティング・システムの目的（マルチプログラミング，タイムシェアリング，あるいはその両者），それが AS(1) あるいは AS(n) であるべきかどうか，写像ハードウェアと管理ソフトウェアの巧緻さ，

† 参考文献 IB 10，p. 41 参照

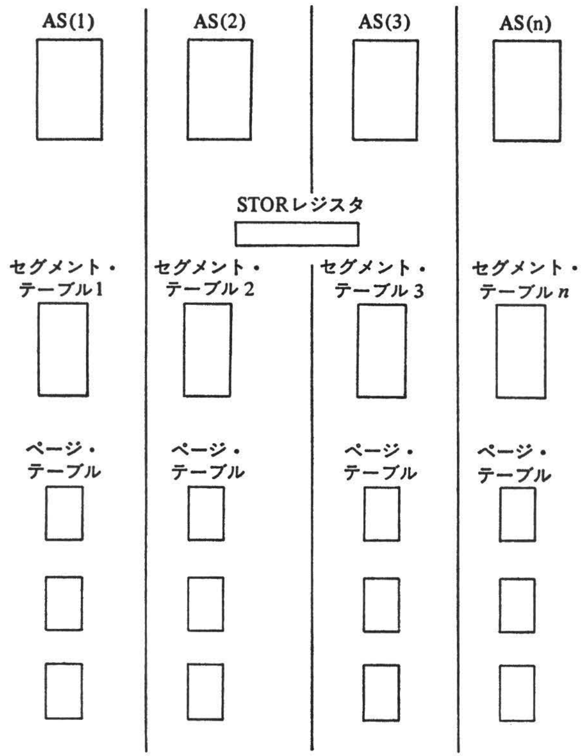


図 11-1.1 AS(n) 写像機構

および利用可能なリソースの速度とサイズの範囲，がある．

CPU のアドレッシング構造が AS のサイズを決める．IBM システム/360/370 および Multics の例が図 11-1.2 に示されている．CPU の実効アドレスに 1 ビット加わるごとに AS のサイズは倍になる．AS の容量に加えて，セグメントの総数とセグメントのサイズもまた考慮すべき重要なことである．AS のサイズが与えられると，少なくともすべてのセグメントが固定の最大サイズを持っているようなシステムでは，セグメントのサイズは 1 つのアドレス空間内のセグメントの総数を決める．(Burroughs の機構はこの点に関してもっと柔軟性がある)．図 11-1.2 に示されているように，Multics は最大のアドレス空間を有している——セグメントは 256,000 に達し，各セグメントは 256 k に及ぶ語から成っている．適切なセグメント・サイズはそのシステムの意図する使用環境とモードに依存して選ばれる．AS(1) では，スケジュールされている

	アドレス・サイズ	AS サイズ	セグメント・サイズ	ページ・サイズ
システム/360	24ビット	16 mバイト	1 mバイト	4kバイト
モデル67	32ビット	4096 mバイト	1 mバイト	4kバイト
システム/370	24ビット	16 mバイト	64kバイト	2kバイトあるいは4kバイ
	24ビット	16 mバイト	1 mバイト*	2kバイトあるいは4kバイ
MULTICS	36ビット	256kセグメント	256k語	1k語

*システム/370のハードウェアは二つのセグメント・サイズを維持している．しかしIBM VSシステムのいずれも1mバイトのセグメントをインプリメントしていない．

(*a*) アドレス・セグメント，およびページ・サイズ

S/360モデル67(32ビット・アドレス)

セグメントI.D.	ページI.D.	バイト・アドレス
—12ビット—	—8ビット—	—12ビット—

S/370 (24ビット・アドレス，64kバイト・セグメント，2kバイト・ページ)

セグメントI.D.	ページI.D.	バイト・アドレス
— 8ビット—	—5ビット—	—11ビット—

MULTICS (36 ビット・アドレス)

セグメントI.D.	ページI.D.	語アドレス
—18ビット—	—8ビット—	—10ビット—

(*b*) 仮想アドレス構造

図 11-1.2　アドレス空間サイズおよび仮想アドレス構造

ジョブに対して，アドレス空間をセグメントの倍数で都合よく割当てることができるので，多数の比較的小さなセグメントを有効に利用する．システム/370の64k-バイトのセグメント(図11-1.2)は，後に示されるように，この使用環境に対して特に適切である．AS(n)システムにおいては，より少数の非常に大きなセグメント(たとえば，1mバイトのセグメント)が適切であろう．そのようなシステムで，セグメントが論理的な要求に基づいたプログラム・モジュールやデータ構造に対して割当てられる場合には，1つの1mバイトのセグメントは潜在的に複数の大きなモジュールやデータ構造を収容しうる．AS(n)システムで供給される大きなセグメントの総数はインプリメンティションで考慮すべきことに影響を及ぼす．利用可能なセグメントより多い論理的なモジュールやデータの要求がある場合には，システムはこれらの論理的な要求を利用可能なセグメントに“押し込め”なければならない．この問題を避けるために，(より高価なハードウェアを持つ)システムは多くの大きなセグメントを各AS

に供給するためにより大きい実効アドレスをインプリメントすることができる．Multics および 32 ビット・アドレッシングのモデル 67 がその例である．これらのいずれを選ぶべきかを考慮する際には，仮想記憶装置は写像構造によって作られるアドレス空間であることを思い出すべきである．システムの作動中，未使用の仮想記憶装置は物理的なシステム・リソースをほとんどあるいはまったく浪費していないのである．それゆえ，Multics のようなシステムは，物理的なシステム・リソースの拘束をうけることなく各ユーザに非常に大きな潜在的な記憶装置を与えることができる．**ページ・サイズ**の効果は 10 章で議論された．小さいページであると内部フラグメンテーションは小さくなり，またおそらくページ置換えも少なくなるが，（主記憶装置内で）より大きなページ・テーブルを必要とする．ページが大きいと，補助記憶装置と主記憶装置の間の転送効率が改善され，ページ・テーブルも小さくてすむ．しかし，ページが大きくなればなるほど，プログラムの現在のワーキング・セットに属しない命令やデータがより多く主記憶装置内に駐在することになる．典型的に通例用いられるページサイズは 4k バイトであるが，これは上の考慮の間の 1 つの妥協である（この選択の正当性を公表しているものはないようであるが）．

仮想アドレスを主記憶装置の番地に翻訳する種々の写像技法は 10 章に述べた．AS がセグメント・ページ構造を持っている場合には，写像過程で，アドレスされた AS の番地のセグメント番号とページ番号が識別されねばならない．それから翻訳はセグメント・テーブルおよびページ・テーブル（あるいは高速連想記憶に保持されたそれらの写しのいくつか）を通じて進行する．セグメント番号，ページ番号およびページ内**偏位**(displacement)として写像装置によって解釈される実効アドレス中のビット数は，ある特定のインプリメンテーションに対するアドレス空間の構造を反映する．図 11-1.2 にいくつかのシステムの仮想アドレスの論理的な構造が示されている．

セグメント-ページの結合は，ページ方式のみの非セグメント機構に比較して，写像テーブルに必要な主記憶装置を減少するのに役立つ．ページ・テーブルは割当てられていないセグメントに対して構成する必要はない．セグメント・テーブルの各エントリ内の存在ビットは，ページ・テーブル内で用いられる存在ビットに類似の方法で，写像装置によってテストされうる．この方法では，割当てられていないセグメントに対して参照しようとするとセグメント・フォールトとなり，これはスーパバイザ・システム・プログラムに対する割込を引き起こす．1 セグメント内でいくつかのページのみが情報を持つ普通の場合には，写像記憶装置でもう 1 つの節約が可能である．セグメント内のすべて

の占有ページが，そのセグメントの(オリジンから始めて)アドレス空間内に隣接して置かれるように拘束されると，使用されているページのためのページ・テーブル・エントリのみがその使用ページ数とともに供給されればよい．この後者の数はそのページ・テーブルに対するセグメント・テーブル・エントリ内に制限数として挿入される．写像の間，ページ・アドレスはその制限に対してチェックされ，もし越えていれば，ページ・テーブル・フォールト割込が発生する．

上の考えを利用した主記憶装置のテーブル空間の節約を示す最もよい例をあげる．AS のサイズを 16 メガバイトとし，64k バイトのセグメントと 4k バイトのページを持つとしよう．ここでは，セグメントあるいはページ・テーブルの各エントリは 2 バイトを必要とすると仮定する．2 メガバイト，すなわち 32の完全なセグメントのみが現在 AS の情報を保持しているとする．これらの条件下で，写像に必要な記憶装置は

32 セグメント・テーブル・エントリ × 2 バイト ＝ 64 バイト
32 ページ・テーブル × ページ当り 16 エントリ × 2 バイト ＝ 1024 バイト
合計写像バイト ＝ 1088 バイト

である．非セグメント・システムでは，単一のページ・テーブルは，実際に占有されている AS がたとえどんなに小さくても，ページ当り 1 つのエントリすなわち 4096 エントリを含むことになるであろうから，写像のために 8192 バイト必要とする．

11-2　AS(1)環境におけるマルチプログラミング ［SM 参照］

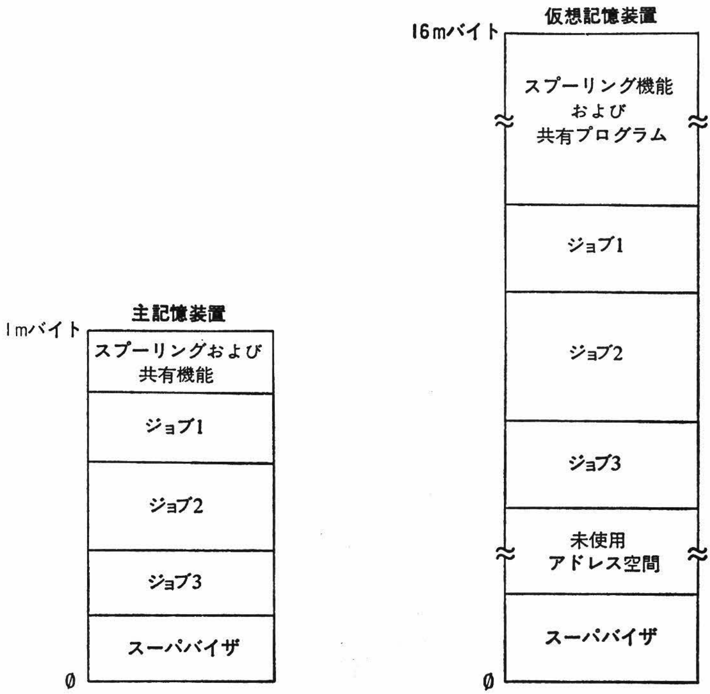
AS(1)型の仮想記憶装置システムは，多くの点で，理解するのが最も容易である．最初に 7 章および 8 章に述べたような通常のマルチプログラミング・オペレーティング・システムを考えよう．AS(1)機構の着想は，ユーザに対して同じ**論理的**構造を維持したまま，ずっと小さな物理的な主記憶装置のかわりに大きな仮想アドレス空間を与えることである．単一の大きなアドレス空間はすべての活動中のユーザ・プログラムとシステム・プログラムによって共有されるが，その様式はそれらのプログラムによって物理的な主記憶装置が共有される慣習的なやり方と同じである．それゆえ，AS(1) システムは，典型的には，通常の(非仮想)オペレーティング・システムと完全に互換性のあるプログラムになるように設計される．

AS(1) は通常のシステムに機能的な類似性があるので，後者の特性のいくつ

かを簡単に要約する.

1. すべてのシステム・プログラムは補助記憶装置, 普通直接アクセス記憶装置上に駐在する.
2. あるシステム・プログラム――すくなくともスーパバイザ――は正規運転の間主記憶装置内に保持される.
3. スプーリングのような, いくつかのシステム機能は, 主記憶装置に駐在するようにするか, あるいは主記憶装置が利用可能な時にスケジュールされるようにする.
4. スケジューリング・アルゴリズムは, 出来るだけ多くのジョブをマルチプログラミングすることによって, ハードウェア・リソースを活動状態に保つよう企てる.
5. 高スループットは最も共通した性能目標である.

非仮想マルチプログラミング・システムの典型的な構造は図 11-2.1(a) に示されている. 主記憶装置はまた全アドレス空間であり, 現用システムでは典型的に 64 k から 4096 k バイトの範囲で比較的小さいので, この貴重なリソースの編成と管理には注意深く気を配る必要がある. 固定サイズの**パーティション**(partition)あるいは可変サイズの**リジョン**(region)機構は典型的にジョブ空間割当に用いられている. いずれの場合にも, システム・プログラマとオペレータは, 主記憶装置のフラグメンテーションを避けるように, 共に走るべきジョブおよびタスクを選択しようと企てる. 主記憶装置は, スプーリング機能, アクセス法のような共有可能なプログラム, およびスーパバイザ・コールやエラー回復ルーチンのようなサービス・プログラムに対して割当てられよう. しかし, このためにゆだねられる空間は, 一般に, 各機能毎に使用の頻度によって正当化されねばならず, またその空間のジョブ処理使用というもう1つの使い道と釣り合いがとられなければならない. 限られた主記憶を同時に共有するジョブとシステム・プログラムの適切なミックスを選択することは, その導入設置での作業負荷, その導入設置の目標(たとえば, スループット, 応答時間, クリティカルージョブ・スケジューリング, など)のような要因に依存し, またもちろん, 観察されるそして常に変化するリソース使用パタンにも依存する. この大部分は, 典型的に, 人間の絶え間ない努力を必要とする. すなわち, システムの活動を観察し, システムのアドレス空間(それはまた主記憶装置である)がどのように割当てられるべきかのスケジューリングの意志決定をしなければならない.



(*a*) 通　常　　　　(*b*) AS(1).

図 11-2.1　AS(1)構造と比較された通常のオペレーティング・システムの構造

図 11-2.1(b) は典型的な AS(1) 仮想記憶装置構成を示しているが，これは，AS(1)がずっと大きなアドレス空間を持っている点を除いては，図 **11-2.1(a)** の通常のシステムと構造的にほとんど同一であることがわかる．ジョブ・スケジューリング，ロード時点でのプログラム再配置，およびプログラム・ローディングのようなシステム機能は変更する必要はない．とはいえシステム・アドレス空間が大きいので性能の改善を目的とする変更はありうる．通常のシステムにおけるのとまさに同様に，ユーザ・プログラムおよびシステム・プログラムは，実行するためには，システムのアドレス空間に駐在していなければならない．しかし，AS(1)では，そのアドレス空間は(主記憶装置ではなく)仮想記憶装置であり，主記憶装置のサイズはオペレータあるいはユーザ・プログラマの関心事というよりむしろ主として性能要因となる．

AS(1) のマルチプログラミング・オペレーティング・システムをインプリメ

ントするには，通常のマルチプログラミングのハードウェアおよびソフトウェアの両者を拡張する必要がある．CPU はアドレス翻訳のための写像装置を持たねばならない．補助記憶装置の一部はアドレス空間を“支援する”ために用いられる．スーパバイザはデマンド・ページング活動を制御する付加的な機能(10 章では**ページング・スーパバイザ**と呼ばれていた)を含む．写像技法については 10 章に述べたので，ここではもっぱら AS の編成と割当て，仮想，主および補助記憶装置間の関係，ページング・スーパバイザ機能，およびシステム・オペレーションのその他の側面について述べよう．

セグメンティションは AS(1) 環境におけるアドレス空間割当てのための便利な手段である．前の節で述べたように，64 k バイトのセグメントを 256 持つ 16 メガバイトの AS は，システム・プログラムおよびユーザ・ジョブに対して割当てうる非常に多数の(比較的)小さなセグメントとなる．スーパバイザ(ページング・スーパバイザを含む)は最初の n セグメントに駐在することになろう(n はスーパバイザのサイズに依存する)．残りのすべてのセグメントはスーパバイザの制御の下に他のシステム機能およびユーザ・ジョブに割当てることができる．**図 11-2.2** は今述べたばかりのセグメント割当て技法を示している．図 11-2.2 の例では，スーパバイザのサイズは 128 k と仮定されている．したがって，セグメント 0 および 1 はスーパバイザに割当てられている．スプーリング・プログラムのような常駐システム機能にたとえば 150 k 必要とすれば，それには 3 セグメント(192 k)が割当てられるであろう．アドレス空間の上部にロードされるとすると，これらの機能は図 11-2.2(b)に示されるように現われてくるであろう．ジョブを処理するために，ジョブ・スケジューリング・プログラムが開始される．動的リジョンを使用するシステムでは，これによって実際上 1 つのリジョンが開始される．スケジューリング・プログラムがたとえば 60 k 要求すれば，図 11-2.2(c)に示すように，1 セグメントのリジョンが割当てられる．もし 80 k のジョブがスケジュールされると，リジョンのサイズは 2 セグメントに増加される．そのときシステムは図 11-2.2(d)に示されるように現われてくる．1 つのジョブ・スケジューラが活動している間に，1 つのジョブ・ストリームが処理される．マルチプログラミングを開始するためには，2 番目のスケジューラが始動されるであろう．通常のシステムにおけるのと同様に，マルチプログラミングの多重度はシステム内の活動中のスケジューラの数に依存する．

アドレス空間の制限しか考慮しないとすれば，AS(1) の環境ではセグメントが利用可能な限りジョブ・スケジューラはいくつも始動しうる．ジョブは隣接

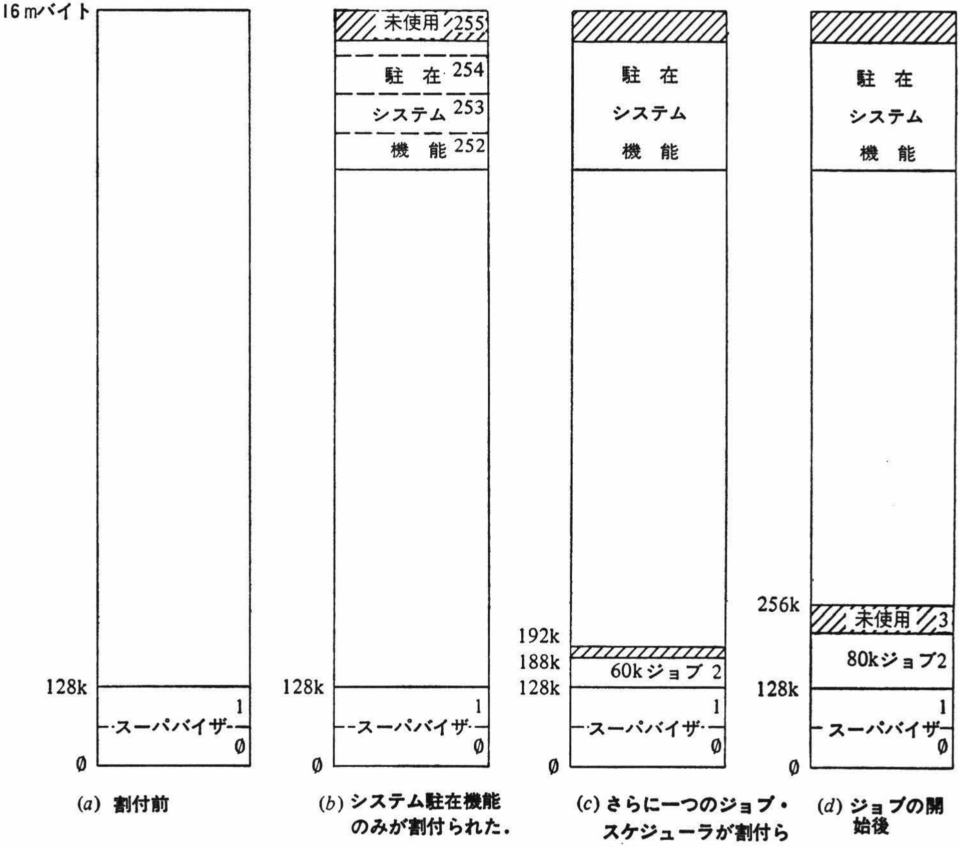


図 11-2.2　AS(1)システムにおけるセグメント割付技法

するセグメントに割当てられねばならない．その結果内部および外部フラグメンテーションが起こることになる．すなわち，セグメントまるまる，およびセグメントの一部が使われないままになりうる．しかし，それに対応する浪費は(物理的でない)仮想的な主または補助記憶装置によるものである．AS(1)機構では，マルチプログラミングの多重度は，典型的には，大きなアドレス空間のサイズによってではなく，むしろ物理的な主記憶装置のサイズ，CPU 速度，および I/O 装置の構成によって制限される．図 **11-2.2** は AS 編成に対する動的リジョン技法を示しているが，パーティション方式システムではパーティションは単純にセグメントのある倍数として定義されてもよいであろう．それか

らAS(1)についてのその他のわれわれの観察のすべてが適用されよう.

AS(1)システムにおける次の関心項目はアドレス空間と主記憶装置の間の関係である. **図11-2.3**はこの関係を示している. スーパバイザのある重大な部分の写しがアドレス空間および主記憶装置の両方のシステム・エリヤに駐在していることに注意しよう. これらは次の理由で主記憶装置内に固定される.

1. スーパバイザはすべての実行中のユーザ・ジョブおよびすべてのシステム・リソースを制御する. ページングによって引き起こされる遅延は性能を耐え難く低下させることになろう.
2. スーパバイザ内の1つのエリヤ, すなわちそれによって制御される作業空間エリヤはマッピング・テーブルを含む. これらのテーブルはアドレス翻訳の間主記憶装置に駐在しなければならない.
3. ページング・スーパバイザはデマンド・ページング・アルゴリズムを用いて残りの主記憶装置を制御する. それは普通非常に頻繁に実行されるので, 満足なシステム性能を維持するために, それは主記憶装置内に固定される.

スーパバイザ内の重大な制御機能しか記憶装置内に固定する必要がない. プログラム・ローディング, スプーリング, エラー回復, スーパバイザ・コール・サービスなどのその他のスーパバイザ機能はアドレス空間の別のエリヤに駐在しうる. 普通それらはページ化される(図11-2.3). デマンド・ページング機構は現在使用中の機能のみが主記憶装置を占有するよう保証すべきである.

スーパバイザが固定された後に残っている主記憶装置はデマンド・ページングのために利用可能なページ・フレームのプールとして役に立つ. システムが走るにつれて, プログラムのアドレス(スーパバイザのも含んで)が翻訳される. ページ・フォールトが起こると, ページング・スーパバイザは参照されたページを主記憶装置内に置かなければならない. それゆえ, **ページ配置**(page-placement)操作は常に要求される. そしてすべてのページフレームが占められると, **ページ置換え**(page-replacement)操作が要求される. 図11-2.3は**全域配置**(global-placement)の戦略を示している. すなわち, 主記憶装置に存在しないページが参照されると, それはいずれの利用可能なページ・フレームにも置かれうる. そのときはすべてのプログラムからのページが1つのページ・フレーム・プールをうばい合う. 置換えが要求されると, すべてのページ・フレームが可能な候補者として置換えアルゴリズムに呈示される. 全域戦略に代

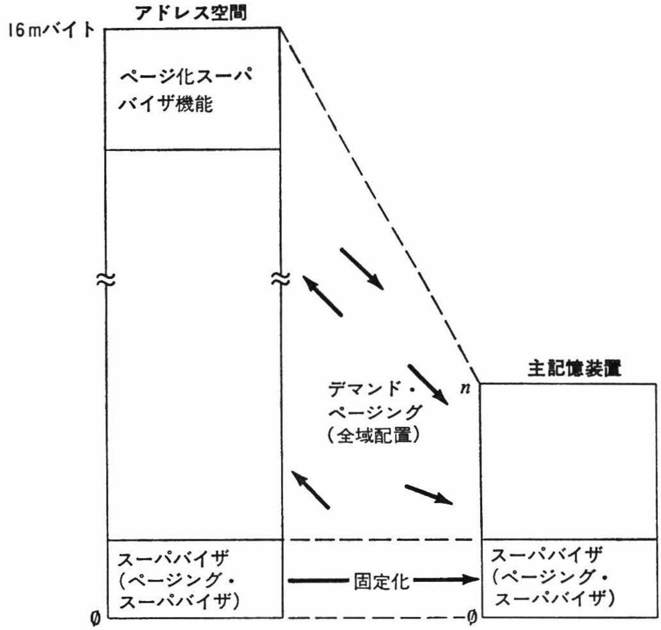


図 11-2.3　AS(1)システムにおけるアドレス空間と主記憶装置の間の関係

わるものとして，スケジュールされたおのおののジョブに対してページ・フレームのサブプールを割当てる**ローカル配置** (local-placement) 戦略がある．特定のジョブに対しては，すべての配置および置換え操作はそのジョブのページ・プール内で生ずることになろう．ローカル配置機構はより良好なシステム保護を提供する――ページング率が高く，したがって局所性が貧弱なジョブが他のジョブと競わず，また他のジョブにまったく影響を及ぼさない――が，そのインプリメンテーションはむずかしい．ユーザは，実行を始める前に，ジョブのページ・プール要求，つまり，その平均ワーキング・セットのサイズを何とかして評価しなければならない．これは本質的にまったく困難であり，また貧弱な評価をするとそのサイズを大きめにしておかなければならないであろう．これらの問題のためにほとんどのシステムは全域配置および全域置換え策を用いる．

ページ配置および置換えの手法は10章で集中的に議論された．ページング・スーパバイザは各ページ・フレームの状態を識別する1つのテーブル(あるいはテーブルの集合)を創成し維持しなければならない．配置操作が必要な時は，

ページ・フレーム・テーブル(1つまたは複数)が利用可能なページ・フレームを識別するために参照される．その後で，ページング・スーパバイザはページ・イン操作を始め，相応するページ・テーブルおよびページ・フレーム・テーブルのエントリを更新することができる．ページ・インが完了すると，プログラムの実行が再開できる．ページ置換えが最初に要求されると，ページング・スーパバイザは置換えのための候補を選ぶためにページ・フレーム・テーブルを参照する．典型的には，LRU 置換えアルゴリズムが用いられ，それはページ・フレーム・テーブルの特別な管理を通じて遂行される．ハードウェアによる援助は普通各ページ・フレームに関連する**参照**および**変更**ビットによって提供される．フレームが参照されるといつでもその参照ビットは ON にされる．フレームの変更ビットが ON になるのはページ内の情報が変更された(情報がストアされた)ときのみである．置換えアルゴリズムの1次候補は，最近参照されなかったページ(参照ビット OFF)，あるいは(参照ビット ON を持つもののうち) 変更ビット OFF を持つものである．すべての参照ビットが ON のときは，それらはすべて OFF にされる．

デマンド・ページングは AS(1) システムでの主記憶装置管理のための通常の技法であるが，例外となる一部類――ある管理機能は主記憶装置に固定される――があることをすでに見てきた．また，ユーザが実行中あるジョブ・ページを主記憶装置のページ・フレームに固定できるような用意がしてある場合もある．いくつかの変形として次のものがある．

1. ある機能，したがってページを，ジョブの実行中主記憶装置内に固定するようユーザが指定する．
2. ジョブ全体をその実行中主記憶装置内に固定するようユーザが指定する．ある小さな種類のジョブはタイミングあるいは I/O 依存性のために，デマンド・ページング環境では実行できない．
3. ある特定のジョブの使用のためにいくらかのページ・フレームを予約するようユーザが指定する．これはローカル配置戦略の1つの変形である．そのジョブはデマンド・ページングを行なう．まずはじめに予約したページ・フレームを用い，それから残りのフレームを他の実行中のジョブと競う．

なおまた，**プリページング**(prepaging)あるいは**コンテクスト・ページング**は，AS(1)でインプリメントされうるもう1つの特徴である．この特徴をもつものでは，ページのある“コンテクスト集合”を主記憶装置内に共存させるようプ

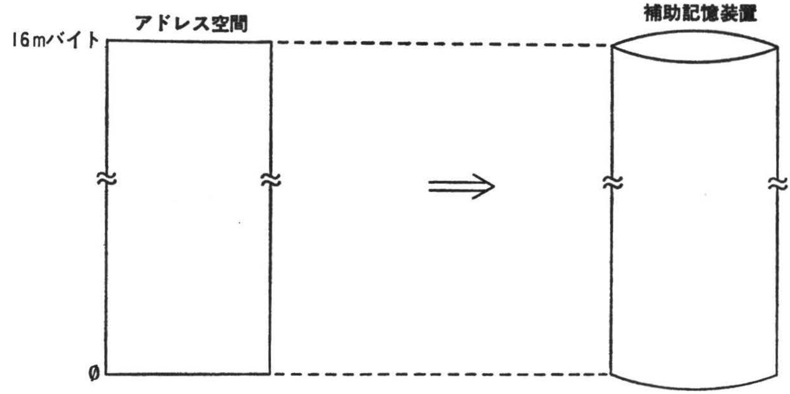
ログラマが指定できる．コンテクスト集合内のいずれかのページに対して初めての参照があると，すべての他の要素が主記憶内にあらかじめページ化される．プログラマがプログラムの全部または一部の参照パタンを知っているなら，そのワーキング・セット（1つまたは複数）を実際上プリページングできることになる．上に述べた各特徴は特殊目的の状況に役立つ．AS(1)環境における実行の標準モードはデマンド・ページングである．

ページ配置および置換え戦略の議論の間，ページ・イン，ページ・アウト活動について言及した．物理的には，これは主記憶装置と補助記憶装置の間の転送を伴う．**図11-2.4**は，AS(1)環境でのページングのためにASと補助記憶装置の間の関係を確立する次のような2つの技法を示す．

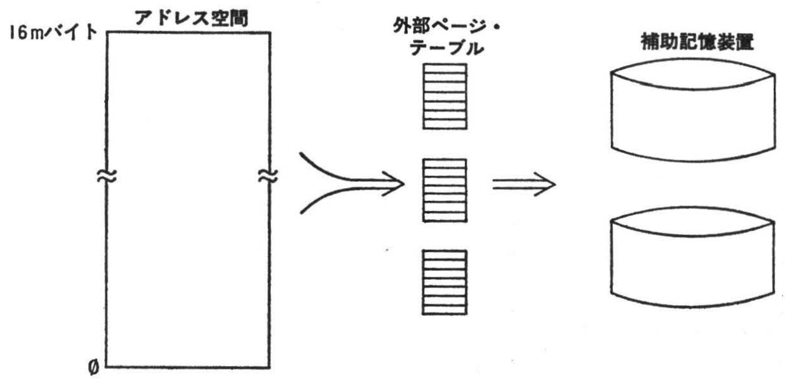
1. 第1の技法，図11-2.4(a)はASのすべてと補助記憶装置の間で巨大（バルク）1対1関係を確立する．
2. 第2の技法，図11-2.4(b)は**外部ページ・テーブル**と呼ばれる写像テーブルを用いる．

10章の序文で，仮想記憶装置は補助記憶装置内の隣接する領域として心に描かれることを示唆した．AS，補助記憶装置の関係に対する図11-2.4(a)に示された巨大1対1対応技法は，まさにそれである．ASのページ化領域は補助記憶装置内の領域に1対1で写像される．外部写像テーブルはまったく必要とされない．ページ・イン操作が要求されると，補助記憶装置のページ・ロケーションは参照されたページの仮想アドレスから決定される．ページ・アウト操作はページをそれらの固定された補助記憶装置のロケーションに返してやらねばならない．巨大対応手法はAS(1)のインプリメンテーションにのみ適している．アドレス空間のフラグメンティションは補助記憶装置のフラグメンテーションに帰する．補助記憶装置の浪費はAS(n)システムでは耐えがたいものとなろう．

10章で述べたように，外部ページ・テーブルはASページ・アドレスを補助記憶装置上の物理的なロケーションに写像するために使用される．各内部ページ・テーブルに対してそのような1つの外部ページ・テーブルがある．ページ・フォールトが起こると，ページング・スーパバイザは参照ページの補助記憶装置内での位置を知るために対応する外部ページ・テーブルのエントリを用いる．ページ・フレームが割当てられた後で，ページ・イン過程を始めうる．置換え操作の間，ページング・スーパバイザは変更されたページを，必ずしもそのページの現在のロケーションでなく，補助記憶装置の任意の利用可能なロ



(a) 手法1　補助記憶装置が全アドレス空間に割付られる1対1巨大記憶装置写像.



(b) 手法2　使用されるアドレス空間のページのみが割付られた補助記憶装置.

図 11-2.4　AS(1)システムにおけるアドレス空間/補助記憶装置の関係

ケーションに対しても“ページ・アウト”出来る．アクセス時間を縮小するロケーション選択アルゴリズムが利用できる．ページ・アウト後，ページング・スーパバイザは補助記憶装置の新たなロケーションを反映するために影響を受けた外部ページ・テーブルのエントリを更新する．外部ページ・テーブルは，ページ・テーブルと一緒に，セグメントが割当てられるときに作成される．その際，補助記憶装置は割当てられていないセグメントに対してあらかじめ確保しておく必要はない．これは AS(1) 環境で価値のある特徴であるが，後で見

るように，それは AS(n) システムではどうしても必要である．

これまで，主記憶装置管理とページングを議論した．ここで2つの他のきわめて重要な操作――システムの開始とプログラムのローディング――についてこれから議論しよう．

IPL の間，スーパバイザは実記憶装置に固定される．IPL の後，システム・オペレータはスプーリング操作を開始し，ページ化された監視機能をロードすることになろう．オペレータはジョブ・スケジューラを開始することによってマルチプログラミングの多重度を制御する．たとえば3つの活動中のジョブ・スケジューラによって，3つのジョブがマルチプログラムされる．

1つのジョブ・ステップがスケジュールされるとき，それは AS の1つのリジョン，および I/O 装置を割当てられる．それから，それは AS に(主記憶装置のフレームにではなく)ロードされる．OS/MVT におけるようにロード時の(静的) **再配置** (relocation)を仮定するなら，ローダ・プログラムはロードされるべきプログラムをデータとして扱い，記憶装置参照をアドレス空間のどこかのリジョンに結びつけるために，そのアドレス定数とシンボル・ディクショナリを修飾† する(図 11-2.5)．プログラム・モジュールにはまたページ・フォー

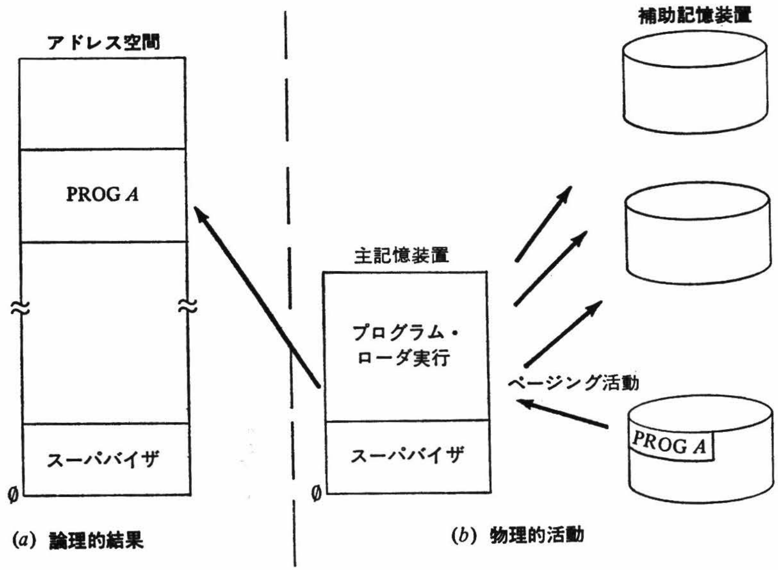


図 11-2.5　AS(1)システムにおける静的再配置プログラム・ローディング

† 参考文献 IB 10, p. 39 参照

マット(page format)が与えられ，処理されたモジュールは補助記憶装置にページ・アウトされる．ASへのローディングが完結した後，制御はロードされたプログラムに移され，それからデマンド・ページング・モードで実行が開始される．もちろん，プログラムのページは利用可能ないずれのページ・フレームにも配置できる．写像やページ・フォールト応答に固有な再配置の過程のことをローディングとは呼ば**ない**ことに注意しよう．ローディングの過程は，プログラムがページ・フォーマットでライブラリにストアされている場合には，単純化することができ，また動的にさえすることができる．**動的ローディング**では，プログラム実行中あるページが最初に参照された時に，写しがプログラム・ライブラリから読まれ，主記憶装置内に配置される．アドレス定数はこの時に再配置される．そのページがその後置換えられる場合には，デマンド・ページング用の補助記憶装置の領域にページ・アウトされる．最初のプログラム・ローディングは，単に，プログラム・ライブラリのページ・ロケーションを反映するために外部ページ・テーブルを作成することを要求するだけである．

ここで，AS(1)が同様な機能を持つ非仮想システムより優れている点を要約しよう．

1. リジョンあるいはパーティションのサイズをずっと大きくすることができ，その結果，ユーザの便利さとジョブ・スケジューリングにおけるシステムの柔軟性をもたらす．
2. より多くの管理機能を“常駐”させることができ，(推定というよりむしろ)実際の要求に従って有効なページングができるであろう．したがって，導入設置において物理的な常駐に関する困難な決定を行なう必要がなくなる．
3. 物理的な主記憶装置はプログラムの真に活動的な部分によってほとんど利用されるであろう．
4. 上記の利点のすべてが，いかなるユーザの，あるいは導入設置での再プログラミングなしで，利用可能である．

11-3 AS(n)仮想記憶装置システム[MA 2 参照]

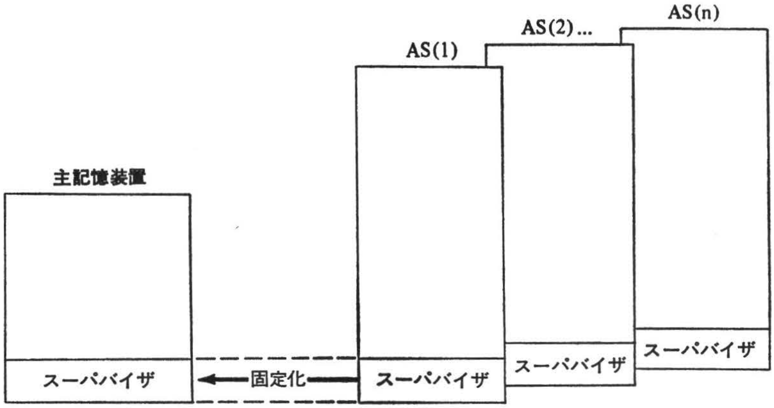
多重アドレス空間を可能にする仮想記憶装置の概念は，11-1節に導入された．そのようなAS(n)のインプリメンテーションの下では，各実行中のユーザは自分自身のアドレス空間を持つ．“ユーザ”はマルチプログラムされているバッチ・ジョブであるかもしれないし，タイムシェアリングのユーザ・セシ

ョン，あるいは多数の端末のネットワークを支えている遠隔処理応用であるかもしれない．AS(n)の機構は，Multics, OS/VS2 Release 2, および VM/370 のようなシステムに対する基本的な設計構造を供給する．ここで，AS(n)の型システムの原理の議論を広げよう．

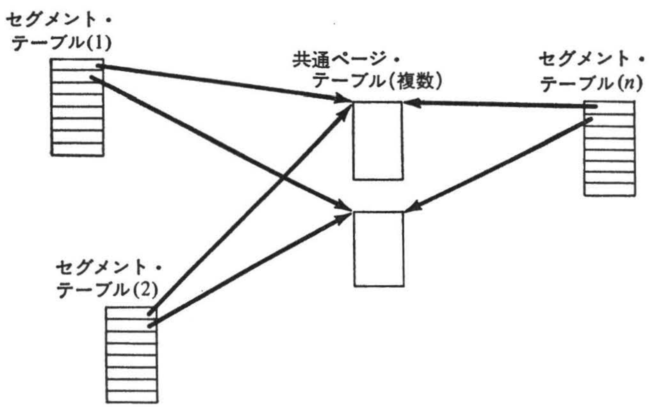
AS(1)型のシステムを越えてAS(n)型のものを選択する理由には次のものが含まれる．

1. 多くの仮想マシン——そのおのおのが物理マシンのすべての論理的性質を持つ——が並行して走ってもよい．
2. スプーリングのようなシステム活動に唯一のアドレス空間を割当てることができる．
3. プログラム・モジュールおよびデータ構造をセグメンテーションを用いてユーザの間で便利に共有できる．
4. 大きなアドレス空間を供給することは，ユーザが指示すべきI/O折りたたみ操作を非常に減らし，より多くの問題に対するより直接的な解を与える．

図11-3.1はある種のAS(n)システムを示している．それには多重アドレス空間があり，スーパバイザは各アドレス空間内の同じセグメントに写像されており，またスーパバイザの重大な部分は主記憶装置内に固定される．スーパバイザは，各ASに共通な一組のページ・テーブルによって写像される．それゆえ最初のASが生成されると，最初のk個のセグメントがそのスーパバイザのために使われる(kはスーパバイザのサイズに依存する)．それから，最初のk個のページ・テーブルがそのスーパバイザを写像する．2番目のASが生成されると，その最初のk個のセグメント・テーブル・エントリは，現に存在するページ・テーブルを指示する．この技法はスーパバイザをその後に続くすべてのASに写像するために使われる．その共通のセグメントは，図11-3.1(b)に示すように，一組の共通のページ・テーブルによって写像される．この写像過程の技法はもっと一般的なセグメントの共有を可能にする．2つの手法が使われる．(1) すべてのアドレス空間に共通な領域を共有可能なプログラムあるいはデータのためにあらかじめ取って置くことができる．(2) プログラム・モジュールおよびデータ構造が，アドレス空間の異なるセグメントで，2つ以上のユーザによって共有されうる．すべてのアドレス空間に共通な領域を予約する，第1の手法は，スーパバイザの部分に対して上に述べたのと同様なものである．それは各アドレス空間で共通のセグメント・テーブルのエントリによっ



(a) スーパバイザは各AS内の同一セグメント内へ写像される.



(b) ページ・テーブルの一つの集合がスーパバイザをすべてのアドレス空間へ写像する.

図 11-3.1 AS(n)システムにおけるすべてのアドレス空間に共通なエリアの共有

て参照される一組のページ・テーブルを要求する. 典型的には, “スーパバイザ・コール” あるいはエラー回復手続のようなシステム・プログラム, 頻繁に使われるアクセス法, コンパイラ, および頻繁に要求されるユーザ関数が, この共通領域を占めることになろう. 各ユーザにとって利用可能なアドレス空間は共有領域のサイズだけ縮小される. 共有される共通のプログラムはリエントラントでなければならないし, 個々のユーザは共有領域に問題プログラムを置

くことができない．この種の手法はIBMリサーチM44，およびIBM VS2/2システムで用いられた．

もっと一般的な共有機構は，インプリメントするのがいくらかむずかしいが，2つ以上のユーザの間に一般的なセグメントの共有を許す．リエントラント・プログラム・モジュールあるいはデータ構造が，2つ以上のユーザ・アドレス空間における異なるセグメントに写像される．その技法は，図11-3.2に示すように，共通領域における共有と類似している．再び，1つの共通ページ・テーブルが2つ以上の異なるセグメント・テーブルのエントリによって参照される．しかし，この場合には，適切なユーザおよびデータ保護を保証する，データ構造へのアクセス権とプログラム・モジュール使用の認定のための申し合せが確立されなければならない．この共有手法では，アクセス権を持つユーザが1つまたはそれ以上の他のユーザによって現在実行されている共有可能なモジュールを参照する場合，現に存在している写しが新しいユーザのASの内に，後者のセグメント・テーブルを通じて，写像される．論理的には，各ユーザは自分のASに1つの写しを持つ．物理的には，ただ1つの写しが主記憶装置内にページ化されており，1つのページ・テーブルと1つの外部ページ・テーブルによって写像される．

前に議論したように，AS(n)環境で生成されうるアドレス空間の数は実際的な理由によってのみ制限される．すなわち，(1) ページングに専用できる補助記憶装置の量，(2) 写像テーブルに利用できる主記憶装置の量，および (3) ページ・フレームのプールの大きさ（すなわち，ページ用主記憶装置のサイズ），

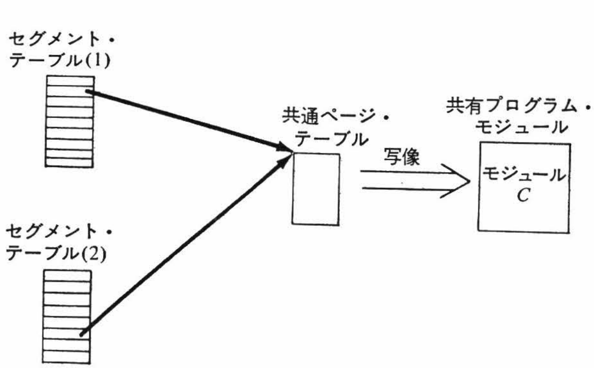


図 11-3.2　2つ以上のアドレス空間の異なるセグメントに写像された共通プログラム・モジュールあるいはデータ構造の共有

である．仮想記憶装置の維持に直接関係する資源は別として，CPU 速度と I/O 装置の利用可能性は，また，AS(n) システムで効率的に走りうるマルチプログラミングとタイムシェアリングの活動度に影響する．性能保護の理由のためにページング・スーパバイザはページング率を監視しなければならない．それを適切に知る方法がスーパバイザ・プログラムに作りつけになっていないと，そのシステムは与えられた主記憶で効率よく処理されうる以上の数のアドレス空間を活動させてしまう可能性がある．そのようなときには，その結果として，各プログラムが(なされる作業がほとんどないままに)そのパラコルにとって十分なページを得ようとするので，非常に高いページング活動，すなわち**スラッシング**(thrashing)と呼ばれる状態になる．AS(n) システムにおけるスラッシングを制御するためのいくつかの尺度が，11.5 節で議論される．

初期の AS(n) のインプリメンテーションはタイムシェアリング環境ではじめて用いられた．IBM TSS(Time Sharing System)はその名でこれを意味している．Multics は同様にタイムシェアリングを強調して設計された．その目的は，端末ユーザに大きな主記憶を持つ汎用計算機の全論理リソースを与えることであった．ユーザおのおのにそれ自身の大きなアドレス空間を与えることはこの目的を大いに助長した．しかし，タイムシェアリング・システムは，仮想記憶装置システム，AS(1) あるいは AS(n) のインプリメンテーションにとって独占的なものでもなく，またその反対でもない．仮想記憶装置はタイムシェアリングおよびマルチプログラミング環境の両者の能力を著しく高めるものである．

11-4 仮想記憶装置システムにおける I/O 操作

チャネルあるいは I/O プロセッサは I/O プログラムを実行するので，それらは真に特殊目的計算機である．通常のオペレーティング・システムでは，チャネル・プログラムおよび I/O バッファ領域は主記憶に駐在し，そこでそれらはチャネル実行中参照される．AS(1) あるいは AS(n) のオペレーティング・システムでは，チャネル・プログラムおよび I/O 領域は仮想記憶装置内に駐在する．しかし，それらはチャネル実行中主記憶装置に駐在していなければならない．当然の結果として，CPU と同様に，I/O チャネルは仮想アドレスを翻訳するための写像過程を必要とすることになる．チャネル・プログラムを写像するための 2 つの方法がある．(1) ハードウェア-ソフトウェア翻訳過程，および (2) 全ソフトウェア翻訳技法，である．

ハードウェア-ソフトウェア・チャネル写像は，各チャネルに写像装置を必

要とするであろう．チャネル・プログラム実行中，すべての参照アドレスは翻訳されるであろう．CPU 翻訳のために使われるのと同様な写像テーブルがまたチャネル写像のために用いることができよう．ページ・フォールトが発生すると，その I/O プログラムを中断し，I/O 操作を再開する前にたとえばページをディスクから主記憶へ移動することによって，そのページ・フォールトにサービスすることが必要となろう．これが，ハードウェア-ソフトウェア写像手法に困難が存する所である．写像装置は別として，チャネルは，I/O 操作を中断し，おそらくページ・フォールト処理中に他の I/O 要求にサービスし，それからその中断された I/O を再開して完了させるためのハードウェアを必要とするであろう．またこの手法には，ほとんどの I/O 装置でのデータ転送がシーケンシャルに行なわれるという性質から起こるタイミングやその他の問題がある．これらの機能にまつわる複雑さと費用はこの手法を用いることを抑制してきた．

全ソフトウェア翻訳技法では，オペレーティング・システムがすべての I/O 写像を行なう．付加ソフトウェアは I/O 管理機能の拡張である．各 I/O 操作に対して，関係するチャネル・プログラムはその操作の開始前に翻訳される．関連する機能には次のものが含まれる．

1. チャネル・プログラムの仮想アドレスを現在の主記憶装置のアドレスに翻訳すること．
2. I/O バッファを含むすべてのページを（もし現在駐在していないなら）主記憶装置にあらかじめ確保し，これらのページを**固定する**こと．
3. それから，その翻訳されたチャネル・プログラムを使うチャネルによって実行される I/O 操作を開始すること．
4. I/O が完了時に I/O バッファを含む固定されたページを解放すること．それから，それらは再びページ置換えアルゴリズムの支配下におかれる．

ソフトウェア翻訳のために必要とされる写像機能は CPU によってすべて行なわれるので，システムのオーバヘッドがいくらか増す．I/O 操作の回数を減らす技法はいずれもこのオーバヘッドを減らす．大きな I/O バッファはより少ない I/O 操作に帰着する．しかし，起こりうるデータ超過を防ぐために，I/O が持続している間，バッファ領域を主記憶装置に固定しておかねばならないし，またバッファ・サイズと利用可能な主記憶装置の間の釣り合いもうまくとらねばならない．普通，プログラムのアドレス空間内にできるだけ多くのデータを置くようにプログラムを設計すると有利である．そのとき，データ

の移動はユーザ向けのファイル処理 I/O というより，むしろシステム向けページング I/O を通じて現われる．

チャネル・ソフトウェア写像の静的な性質は，ページング環境で処理される I/O のプログラミングの実際の型式にある制限を置く．もしチャネル・プログラムが **I/O 実行中**に動的に（たとえばバッファ領域に対する）アドレス・ロケーションを修正すると，ソフトウェア事前写像過程によってそれを前もって検出できない．その結果，主記憶内の間違ったロケーションへ(あるいはそれからデータを)転送しようとする企てが生ずるかもしれない．そのような企てはチャネルの記憶装置保護機構によって検出し，流産させてしまわなければならない．しかし，動的チャネル・プログラム修正は，いくつかのきびしい制限のもとに許されうる．そのプログラムがロードされる主記憶装置の領域と同じアドレスを持つ仮想記憶装置の領域にそれを割当てなければならず，またそのプログラムを全実行期間固定しなければならない．通常の(非仮想)システムでは，チャネル・プログラムを動的に修正する主な理由はバッファ領域に使われるアドレス空間を維持することであった．アドレス空間が主記憶装置である場合には正当な考慮であるが，このことは仮想記憶装置システムでは重要ではない．チャネル・プログラムの修正を許す必要性は，非仮想システムのために書かれた，チャネル・アドレスを自己修正するプログラムの互換性を維持することを望む場合に，主として起こってくる．

11-5 マルチプログラミングとスラッシング [BR, BU, DE 2 参照]

マルチプログラミング・システムでは，多くのプログラムが主記憶装置の空間を共有している．全域ページ置換え戦略を用いる仮想記憶装置環境では，これらのすべてのプログラムが，ページ・フレームの単一のプールに対して競合している．実行中のプログラムはそれぞれ，前に述べたように，記憶装置の実行参照パタンに依存するワーキング・セット(パラコル)サイズを持つ．これらのワーキング・セット・サイズの複合したものは**システム・ワーキング・セット**のサイズとみなしてよいであろう．これがページ・フレーム・プールにおける記憶装置のサイズを超えるときは必ず，**スラッシング** (thrashing) と呼ばれる一種の不安定性が生ずる．その原因は主記憶装置内のプログラム間でのページ・フレームの激烈な奪い合いであり，それらがそれぞれのパラコルを拡張しようとしているからである．簡単な例として，たとえば2つのプログラム，X と Y を仮定する．それらのページは当座はすべての利用可能なページ・フレームを使用するが，どちらのプログラムもそのパラコルにないとする．現在 X

が走っているとする．それはそのパラコルを下まわっているので，それが実行しなくなる確率が高い．というのは遠からずそのアドレスの1つがページ・フォールトを発生するからである．主記憶装置は完全に占められているので，1ページが置換えられねばならない．ページング・スーパバイザは，置換えるべき“最良”のフレームが現在 Y のページの1つを保持しているのを見い出したとする．ページング過程を開始後，X はページ・インが完了しなければ進みえないのでシステムはおそらくCPUをプログラム Y に割当てるであろう．しかし，Y は，X のページ・フォールトを満たすために，Y のページの1つを事実上 X に与えているので，一瞬前よりいっそう大きい度合でそのパラコルを下まわっていることに注意しよう．それゆえ，高い確率で Y はごく短時のうちにページ・フォールトに出くわし，その時点で，X と Y の役割の逆転を除いて前と同じ状況がまた存在することが予期できよう．限られた数のページ・フレームに対するこの競合は，プログラムがそれぞれのパラコルに達しようと実りのない努力に奔走して，非常に高率のページ・フォールト，低CPU利用率，そしてそれゆえほとんど無効な作業という結果になる．

スラッシングを検出する手段は想像するのに困難でない．第1に，システム・プログラムがページ・フォールト率とCPUアイドルを監視し，これらが(ある予め設定した閾値を超えて)異常に高いことを検出することは比較的容易である．しかし，スラッシングは容易に検出できるが，次のような特定の理由を決定するのは困難である．

1. 全実行負荷，すなわち，実行中のユーザ・ジョブとシステム・プログラムの数とサイズは，システムの主記憶装置のサイズにとって不当なものである．
2. 特定のジョブ(1つまたは複数)が“悪い”参照パタン，すなわち大きなワーキング・セットを持っている．

作業負荷の知識に基づく判断やいくつかの運転経験から，導入設置での主記憶装置容量にとって合理的な全負荷を推定できる．生産的なランに先立つ特定のプログラムの局所性の測定は (10章で議論されたように) 別の問題である．導入設置でスラッシングを防ぐための予防措置としてプログラムの局所性をあらかじめ測定しようとすると，人の努力とマシン・タイムの結果的なコストは高く禁止的なものとなろう．また，局所性を測定するために設計された**道具**(tool)は一般には利用できない．ページング・スーパバイザが全システムのページング率に対する実行中の各ジョブの寄与を動的に決定しようとすると，実

行中のオーバヘッドは禁止的なものとなろう．これらの理由から，よく報告されてきた基本的なスラッシングのアルゴリズムはどのジョブあるいはプログラムがスラッシングにもっとも責任があるかを区別しない．スラッシングの防止はシステム性能の保護にとって重要である．この目的のためのアルゴリズムは**スラッシング・モニタ**あるいは**ロード・レベラ**(load leveller)と呼ばれている．

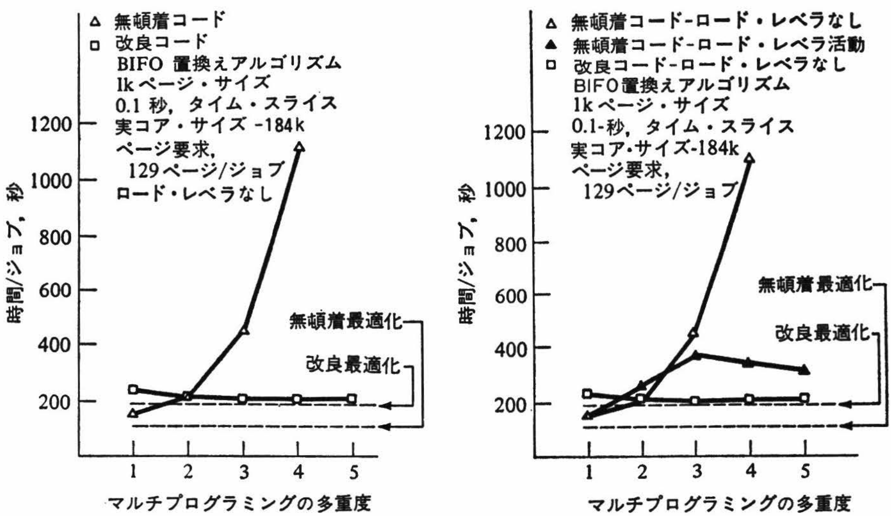
AS(1) システムでは，最も単純なロード・レベリング(負荷を平均化すること)技法は，システムのページング率が特定の閾値を超えたとき，優先度が最も低いジョブ(1つあるいは複数)を不活動の状態にすることである．その閾値はシステム設計の際に固定してもよいし，あるいは，導入設置の際に直接・間接的に指定されてもよい．ジョブの**不活動化**が起こったら，不活動ジョブの主記憶装置内のすべてのページが，ページング・スーパバイザの制御の下で補助記憶装置にページ・アウトされる．その後で，この手続によって取り戻されたページ・フレームは残りの活動中のジョブおよびページ化システム・プログラムにとって利用可能となる．システムは，もちろん，不活動ジョブを後に再び活動状態にすることによってそれらを保護しなければならない．また，あるマルチプログラミングの多重度を最良のスループットに維持するために，ページング・スーパバイザはページング率を継続的に監視し，可能なときには不活動ジョブの再活動化を試みる．スラッシングを防止し，そしてジョブを再活動化するためのこの監視過程は，システム運転にいくらかのオーバヘッドを付加する．不活動化および活動化の操作があまりに頻繁に試みられるならば，結果としてのスループットは，マルチプログラミングの最大多重度をより低くしたAS(1) システムを運転するよりも，悪くなることもありうるであろう．AS(1) システムはスラッシングを防止して有効な仕事をさせることができるが，導入設置の際，マルチプログラミングの多重度を非現実的に高くしようと企てるべきではない．

スラッシングを防止するためのロード・レベリング技法の有効性は，実験的 IBM M44 システムのためのシステム・プログラムの設計者である R. O'Neil によって実証された．プログラミング様式，マルチプログラミングの多重度，ロード・レベリング，主記憶装置のサイズ，および，その他の要因の間の関係を調べるための注意深い対照実験が Brawn および Gustavson によって報告された．単一プログラム環境における彼らの実験は10-7節で議論された．マルチプログラミングの試験では，同一のソーティング・プログラムの n 個の写しがシステムの負荷であった．ジョブあたりの時間(スループット)が性能の

尺度として選ばれた．予想通り，マルチプログラミング環境におけるプログラミング様式の効果は単一プログラムの場合におけるそれらと類似していた．図11-5.1(a)で見られるように無頓着にコーディングされたプログラムに対しては，2つのプログラムが同時に走る場合においてさえ，すでにある程度のスラッシングが認められる(ジョブあたりの時間が単一プログラムの場合より劣っているから)．マルチプログラムの多重度が増加するにつれてスラッシングも増加し，4つのプログラムが競合している場合にはそれはまったくひどいものになっている．また同図は，“最も改良された”ソート・プログラムのマルチプログラミングの効果も示している．この場合には，5つまでのプログラムのマルチプログラミングに対してはスラッシングがないことがわかる．

ロード・レベラ(プログラムを不活動状態にするスラッシング・モニタ)の付加の効果が図11-5.1(b)に劇的に示されている．比較のためにロード・レベリングがない場合もまた図示されている．これらの研究の結果は次のように要約できよう．

1. スラッシング制御は実現可能であり，有効である．



(*a*) マルチプログラミング環境におけるジョブ・スループットに及ぼすプログラミング様式の効果：ソーティング(10,000の10語アイテム)

(*b*) ジョブ・スループットに及ぼすロード・レベラの効果：ソーティング(10,000の10語アイテム)

図 11-5.1　AS(1)システムにおけるマルチプログラミングとロード・レベリング

2. ロード・レベラは，貧弱なプログラミング様式にもとづくような，貧弱なプログラムの局所性を完全に補償できない．
3. ロード・レベラは，ジョブのいくつかの組み合せがシステム性能を不当に劣下させ，その結果他のジョブが合理的な時間内に完了できなくなるというようなことが起こらないようにすることによって，ある程度の性能保護を保証する助けとなりうる．

(ここに示されていない)その他の実験は，単一プログラミングの場合におけるように，マルチプログラミング・システムにおけるページ置換えアルゴリズムはスループットの性能の主な要因ではないことを示している．

ロード・レベラ・プログラムのジョブ不活動化技法は，スラッシングによるスループット低下の防止に有効であることが上でわかった．しかし，タイムシェアリング・ユーザにもまたサービスしている仮想記憶装置システムにおいては，不活動化は応答時間を耐え難いものにしてしまうことがある．AS(n) システムはバッチ作業負荷に加えて典型的にたとえば40の接続されたタイムシェアリング・ユーザにサービスしている．もし数多くのタイムシェアリング・ユーザがデマンド・ページングのみに基づいて主記憶装置を競合するとなると，ページング率はかなり高くなりうる．この理由のために，いくつかの仮想記憶装置タイムシェアリング・システムは**ブロック・ページング**技法を用いる．それによると，ユーザ・プログラムの活動化にはユーザのプログラムの多くのページのスケジューリングと移動が伴う．これは(9章に議論されたような)1種のスワッピングであるが，通常のシステムにおけるスワッピングよりはるかに効率的である．1つには，仮想記憶装置システムは，プログラムのかなり活動的な領域しか関係しないように保証しているからである．あるジョブの大半がページ・アウトされなければならないときに，最後の“スワップイン”以後に変更のあったページしか補助記憶装置に移動する必要がないからである．通常のシステムでは，プログラムの占めたアドレス空間のすべて(最後のスワップイン以後参照されなかった部分を含んで)が補助記憶装置にスワップされる．ブロック・ページング戦略はタイムシェアリング仮想記憶装置環境において容認可能な応答時間を提供する．もちろん，たとえブロック・ページングが用いられたとしても，ページング率があまりに高くなった場合には，システムはそれ自身を保護しなければならない．1つの戦略はいずれの新たなユーザに対しても“ログ・オン”を拒否することである．

スラッシング・モニタの動的操作にはいくらかのシステム・オーバヘッドが

伴う．これは，その作業負荷特性の観察と測定に基づいた導入設置計画によって，縮小されるであろう．そのような情報は，種々のジョブ・ミックスに対して試みられるマルチプログラミングの最大多重度を制御するために使用されうる．また導入設置に際して，仮想記憶装置環境に対するプログラミング様式を改善するために，簡単な指針に従うようプログラマを促すべき多くのことがある．

11-6　IBM 仮想記憶装置(VS)システム
［AU 2, IB 1, IB 2, IB 3, IB 4, IB 5, IB 6, SC 参照］

1972年中期に，IBMはそのシステム/370のほとんどのモデルに対して仮想記憶装置を発表した．写像ハードウェアはその当時現存したほとんどのシステム/370モデルに付加され，また2つの新モデル（モデル158, 168）にはそれが含まれていた．4つの新しい仮想記憶装置システムが，これらの新しい構成を支えるために製作された．この節では，その新しいオペレーティング・システムの特性の手短かな要約から始め，システム/370の写像ハードウェアの記述に進み，それからIBMオペレーティング・システム/仮想記憶装置2（OS/VS 2)を記述しよう．

IBM VSシステムの設計は，システム/360に対して開発されたIBMオペレーティング・システム（OS）およびディスク・オペレーティング・システム（DOS)に，基礎を置いている．互換性を高めるために，DOS/VSおよびOS/VSシステムに対してAS(1)型のインプリメンテーションが利用されたが，そこではシステムのアドレス空間として主記憶装置を仮想記憶装置で置換えている†．これは図11-6.1に示されている．システム/370ハードウェアは2つのページ・サイズ2kおよび4kを可能としていることに注意しよう．小型および中間のマシンのモデルに対して意図されたVSシステムは2kページ・サイズを用いるが，その結果，理論上，より小さなワーキング・セットになる．すべてのVSのインプリメンティションは，64kセグメントと24ビット・アドレッシングを支援している．

すべてのシステム/370仮想記憶装置モデルは，写像用に，共通の論理的な動的アドレス変換（DAT）機能を有している．しかし，DAT機能をインプリメントするために必要とされるハードウェアは，速度-コストのトレード・オフのためにモデル間で異なる．すべてのモデルで実効アドレス・サイズは24

† これらのシステムについてのより多くの情報は参考文献 IB 10, pp. 49-83 に含まれている．

システム	インプリメンティション	先行システム	アドレス空間サイズ	セグメント・サイズ	ページ・サイズ	S/370 モデル
DOS/VS	AS(1)	DOS	16 m バイトまで	64 k	2 k	小型から中間
OS/VS 1	AS(1)	OS/MFT	16 m バイトまで	64 k	2 k	中間から大型
OS/VS 2-Rel 1	AS(1)	OS/MVT	16 m バイト	64 k	4 k	中間および大型
OS/VS 2-Rel 2	AS(n)	OS/MVT	各ユーザごと	64 k	4 k	大型
		あるいは	16 m バイト			
		OS/VS 2-Rel 1				
VM/370	AS(n)	CP/67	各ユーザごと	64 k	4 k	中間および大型
			16 m バイトまで			

図 11-6.1 IBM システム/370 仮想記憶装置オペレーティング・システム

ビットであるから，その結果，16 m バイトのアドレス空間となる．DAT は，セグメント-ページのアドレス空間編成を供給する．しかし，セグメンテーションの程度は各 VS システムで異なる．図 11-6.1 に示すように，いずれの VS オペレーティング・システムも 1 m バイトのセグメントを利用していないが，ハードウェアは 64 k あるいは 1 m バイトのいずれかのセグメント・サイズを可能としている．マシンの制御レジスタは使用されているセグメント・サイズとページ・サイズを示す．このレジスタは特権モードで実行しているプログラム，この場合は，VS 制御プログラム，によってのみ変更されよう．セグメント-ページ構造のアドレス空間に対して 10 章に述べたように，実行中，アドレス翻訳(写像)が起こる．通常の運転におけるのと同様，実行アドレスはシステム/370 **ベース偏位**(base-displacement)アドレスから生ずる．それから実効アドレスは DAT 機能によって翻訳され，その結果は参照アドレスの主記憶装置上のロケーションになるか，ページ・フォールトになる．STOR レジスタ機能はシステム/370 制御レジスタによって供給される．セグメント・テーブルおよびページ・テーブルは VS オペレーティング・システムによって作られ，維持される．システム/370 は，アドレス翻訳のない通常のモードで働くか，あるいは，VS オペレーティング・システムに必須な DAT 機能を用いて働

く．DAT 機能は特権命令によって ON か OFF に切換ることができる．

機能的には，DAT 過程はすべてのシステム/370 モデルで同じである．アドレス翻訳はテーブル・ルックアップ手続（主記憶装置内に駐在するテーブルを用いて）によって行われる．特別の高速翻訳装置は，実行速度にほとんどあるいはまったく損失を与えることなく，すべての最近に参照された仮想アドレスを翻訳する．物理的には，小型と大型のシステム/370 の間で DAT 機能のインプリメンテーション上の差異がある．より小型のモデルは DAT 機能をインプリメントするためにほとんど**マイクロコード**(microcode)を用いる．10 章に述べられたものと同様な，小型の連想記憶が高速翻訳のために用いられる．より大型のシステム/370 モデルは実行速度を増加するために DAT 機能のインプリメントにハードウェア回路を用いる．最近参照されたアドレスの翻訳は**翻訳ルックアサイド・バッファ**(Translation Lookaside Buffer；略して **TLB**)と呼ばれる高速バッファ記憶内で行なわれる．システム/370 のチャネルは DAT 機能を持たない．チャネル・プログラムは，11-4 節に述べたように，ソフトウェア技法によって写像される．

3つの新しい IBM VS オペレーティング・システムの中では，オペレーティング・システム/仮想記憶装置 2 (OS/VS2) が最も大きく，また最も多くの機能を含んでいる．OS/VS2 は，中間-大型システム（モデル 145 から 168）のために設計され，OS/MVT（8章参照）に取ってかわることを意図している．OS/VS2 のインプリメントには 2 つの型があるが，それらはかなり異なっている．

VS2　Release 1 (VS2-Rel 1) は AS(1) のインプリメンテーションであり，OS/MVT に取ってかわるものとして開発された．

VS2　Release 2 (VS2-Rel 2) は大型のシステム/370 で用いられることを意図した AS(n) システムである．

OS/VS2 の議論を望むいくつかの理由がある．

1. 機能に関して，VS2 とその先行システム OS/MVT を対比させること．
2. 比較の基礎として，MVT から始めて仮想記憶装置をインプリメントするために必要な付加すべきソフトウェアを識別すること．
3. 仮想記憶装置によって OS に付加された性能の潜在力を調べること，およびこれと VS2 のインプリメンテーションに対して必要とされる追加リソースを対比すること．

最初に VS2-Rel1 を議論し，それから VS2-Rel2 において異なっている機能を言及しよう．

図 11-6.2 に，MVT のアドレス空間の構造が VS2-Rel1 と比較されている．2つのシステムの構造の違いはほんのわずかである．最も重要な差はアドレス空間のサイズである．MVT における物理的な主記憶装置のサイズに比較して，VS2 における仮想記憶装置は 16m バイトである．VS2 における大き

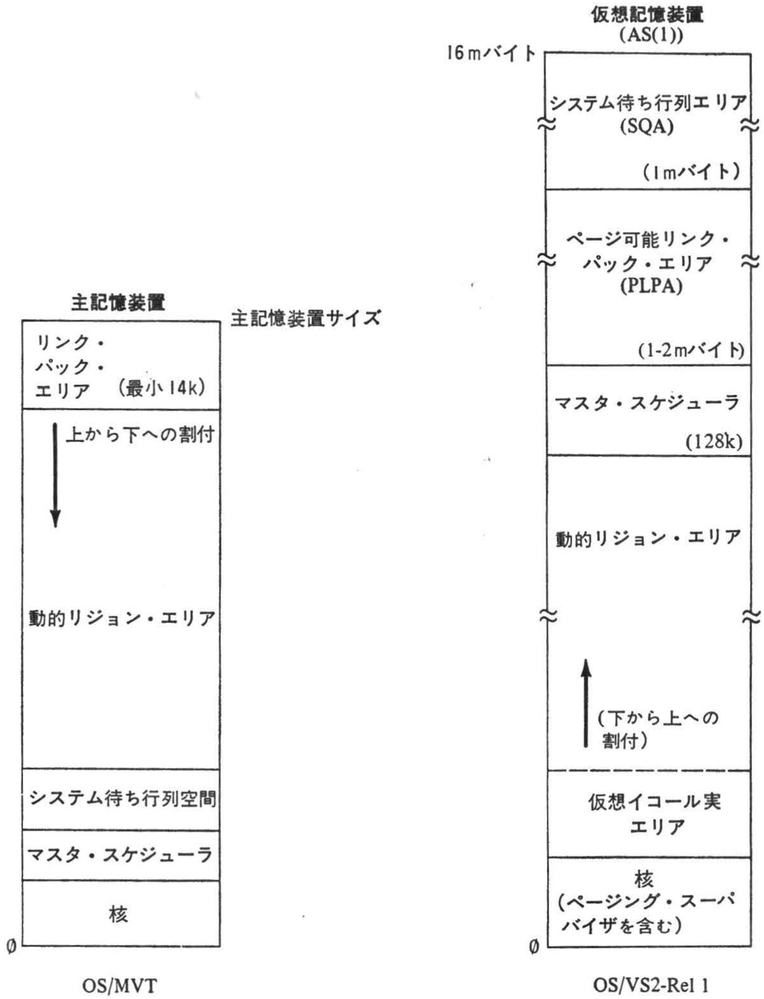


図 11-6.2　OS/MVT および OS/VS2-Rel1 の AS の内容と構造

な AS のために，かなり大きいセグメント(64 k)が空間割当ての基礎として用いられる．したがって，SQA，PLPA，ユーザ・リジョンなどは，64 k の倍数である．セグメント・テーブルは，アドレス翻訳のために利用する他，セグメント割当て状態を指示する．

VS2 の主な利点はその大きなアドレス空間に由来する．たとえば，**システム待ち行列エリア**(System Queue Area；略して **SQA**)には，残りのシステム構造にほとんど影響を及ぼすことなく，AS の 1 m バイトを与えることができる．SQA は VS2 システムの待ち行列と，テーブルを保持するために VS2 の作業エリアとして使われるが，それには(動的リジョン・エリアを除く)すべての仮想空間を写像するセグメント・テーブルとページ・テーブルも含まれる．SQA のサイズは 1 m バイトであるが，主記憶装置に駐在するのは割当てられた SQA のページのみである．VS2 の**ページ可能リンク・パック・エリア**(Pageable Link Pack Area；略して **PLPA**)はすべての SVC ライブラリを含む．SVC は VS2 ではデマンド・ページ化され，そして(MVT で見られたような)主記憶装置遷移エリアの必要性が削除される．デマンド・ページング戦略の利用の結果として，頻繁に使われる活動的な SVC は実際上主記憶装置にとどまる傾向がある．また I/O アクセス方式も PLPA 内に置かれて，2 つ以上の活きたユーザによって共有されてもよい．事実，共有可能なリエントラント・プログラムはいずれも PLPA に置かれうる．PLPA のサイズはかなり大きく，おそらく 2 m バイトになりうるが，しかしその容量はそれらのデマンド・ページング活動に従ってのみ主記憶装置の利用率に影響を及ぼす．PLPA のほか，マスタ・スケジューラおよびスプーリング機能もまた VS2 ではページ化される．MVT に比べて空間的な制約が少ないので，VS2 にとって必要で望ましいシステム機能についてより深い考慮が払われてきた．

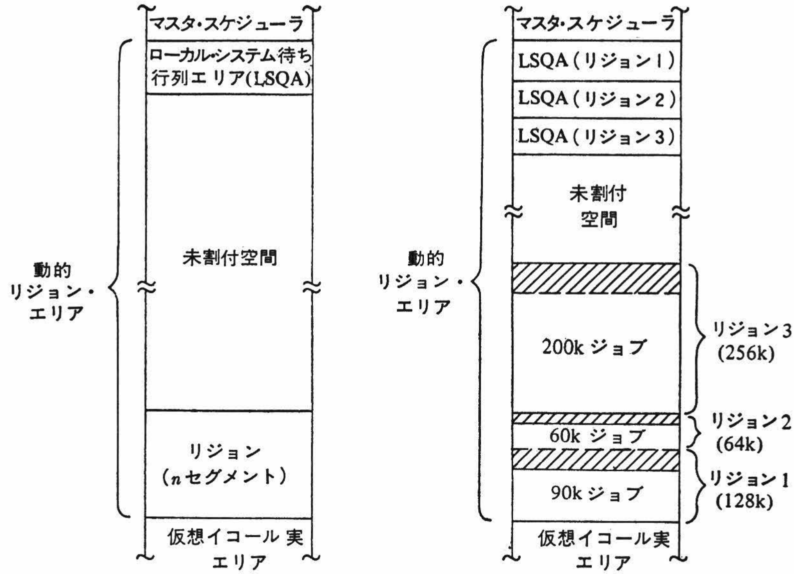
VS2 および MVT において，**核** (nucleus) あるいはスーパバイザは同一の機能，すなわちシステム・リソースと基本的なシステム活動の制御を遂行する．しかし，VS2 では，主記憶装置と I/O 活動の制御は，デマンド・ページングのために，より複雑である．それゆえ，**ページング・スーパバイザ**を付加するため，また入力出力スーパバイザ(IOS)にはチャネル・プログラム翻訳機能が必要となるため，VS2 における核のサイズはより大きい．VS2 ページング・スーパバイザは，システムのページ・テーブルを維持し，ページ・フォールトに対処し，そしてページ配置およびページ置換えアルゴリズムを用いて主記憶装置の割当てを制御する．VS2 は，LRU 置換えアルゴリズムのインプリメンテーションを用いて，現在参照されているページを主記憶装置に保持しよ

うとする．ページ装置は全域的な基礎に基づいて行なわれる．すなわち，どのユーザあるいはシステム機能がページ・フォールトを引き起こそうとも，利用可能ないずれのページ・フレームも配置選択の候補となる．

システム/370のチャネルは翻訳ハードウェアを持っていないので，VS2のIOSは，11-4節に述べたように，チャネル・プログラムの翻訳を行なわなければならない．VS2の核はきわめて重大な制御機能を遂行するので，その核をシステム運転中主記憶装置内に固定することが必要となる．核の実行中にページ・フォールトによって引き起こされる遅れを大目に見ると必ずシステム全体の効率低下を招くであろう．

同様にまた，VS2は，ページ・フォールト遅れを許容できないユーザのジョブ・ステップをその実行中主記憶装置に固定できるようにする機能を提供する．それは**仮想イコール実**(Virtual Equals Real; **V = R**)オプションと呼ばれる．主記憶装置のサイズの範囲内に入るVS2のアドレス空間の一部が，このオプションのために，核の上部にあらかじめ確保される(図11-6.2参照)．ユーザが1つのジョブ・ステップをV = Rと(JCLパラメタで)指定したとき，対応する主記憶装置のロケーションにそのジョブ・ステップをロードし固定することができれば，VS2はV = Rエリアからそのジョブ・ステップに対して空間を割当てるであろう．このジョブ・ステップは，ページの競合あるいはチャネル・プログラム翻訳のオーバヘッドなしに実行できるという長所を持っている．V = Rジョブに対するこれらの有利は他のジョブに対する不利に帰する．すなわち，そのシステムの残りのものにとって利用可能なページ・フレームが縮少する．きわどいタイミング依存性を持つジョブ・ステップやIOSチャネル翻訳機能によって翻訳できないI/O機能はV = Rの取扱いを必要とする．

標準的には，VS2ジョブはデマンド・ページ化され，図11-6.2に示すように，動的リジョン・エリアの底の方からVS2リジョン内にスケジュールされる．リジョン構造は図**11-6.3**(a)に示される．アドレス空間は，底から上にむかってセグメントの倍数で，ジョブに割当てられる．各リジョンは**ローカル・システム待ち行列エリア**(Local System Queue Area; 略して **LSQA**)のために1あるいはそれ以上のセグメントを使う．LSQAは実行中の特定のジョブに属する待ち行列とテーブルのための作業空間として使用され，またそのリジョンに対するページ・テーブルを含んでいる．LSQAセグメントは上から底にむかってリジョンに対して割当てられる．図11-6.3(a)は，1つの活動リジョン(1つの活動ジョブ・スケジューラ)を持つVS2の作用を描いている．ある



(*a*) VS2-Rel 1内のリジョンはセグメントの倍数でジョブに割当てられた空間である．それに加えてリジョンLSQA空間のために少くとも1セグメントある．

(*b*) 三つのジョブがマルチプログラミングされている場合の，VS2-Rel 1における動的リジョン・エリアの配置の例．(斜線で示された)仮想記憶装置のフラグメンティションは未使用リジョン空間から生ずる．

図 11-6.3　VS2-Rel 1 におけるリジョン構造と割付

ジョブが 90k を要求すれば 2 セグメント(128k)が割当てられる．150k を要求すれば 3 セグメント(192k)となる．**図 11-6.3(b)** は，90k, 60k および 200k のジョブを仮定した場合の 3 つの活動リジョンに対するアドレス空間の割り付けを示す．各リジョンは 1 セグメントの LSQA を持つ．セグメントの割当て単位はフラグメンテーションの原因となるが，そのフラグメンテーションの結果による浪費は仮想記憶装置内においてであり，主記憶装置内においてではないことに注意しよう．

VS2-Rel 1 におけるジョブ・スケジューリングは MVT によく類似している．ジョブには JCL を用いてクラスと優先度が割当てられ，そして各ジョブ・クラスに対してスケジューリング待ち行列がある．システム・オペレータは，マスタ・スケジューラを通じて，イニシエーター-ターミネータ・プログラ

ム(ジョブ・スケジューラ)を開始し，1つのイニシエータに対して1つまたはそれ以上のジョブ・クラスを割当てる．活動中のイニシエーターターミネータのおのおのは1つのリジョンと関連している．したがって，システム内のマルチプログラミングの多重度は活動中のジョブ・スケジューラの数によって決定される．VS2ではユーザ・ジョブに対する記憶装置保護は，記憶装置保護キーによって実現されているのではなく，むしろシステムのセグメント・テーブルのエントリ内の保護ビットによって実現されている．PSW内の保護キーが0(スーパバイザ状態)のとき，核は仮想記憶装置のすべてにアクセスしうる．このことは，**スーパバイザ・セグメント・テーブル**と呼ぶ2つのセグメント・テーブルの1つをアクセスすることによって，行なわれる．PSWキーが0でない(問題プログラム状態)のときは，VS2はアドレス翻訳のために第2のセグメント・テーブルを用いる．実行中のユーザに属するセグメントのみがそれらの保護ビットをOFFに変えれる．他のセグメントに対してアクセスしようとすると“保護”割込が生ずる．記憶装置キーが保護のために用いられていないので，VS2はMVTのように，15リジョンに限られていない．VS2の限界は256リジョンである．したがって，VS2における16mバイト・アドレス空間が，マルチプログラミングの多重度に対して課す論理的制限はたとえあるとしてもほんの少しである．しかし，主記憶装置のサイズとCPUの能力はマルチプログラミングの実際の多重度を制限している．あまりにも多くのイニシエータを開始すると，結果的にスラッシング，それに続いてVS2のスラッシング・モニタによるジョブの不活動化が起こるであろう．そしてそれはページング率が生産的水準にまで和らぐまで続くであろう．

VS2におけるTSO機能はTSO/MVTと同一である(9章参照)．しかし，仮想記憶装置環境では異なるインプリメンテーションを考慮している．MVTでは，TSOのユーザ・エリアは主記憶装置と補助記憶装置の間でスワップされている．VS2-Rel1ではページングが利用されている．TSOのユーザが“ログオン”すると，彼はMVTと全く同様に1つのリジョンに割当てられる．実行中，TSOユーザ(複数)はデマンド・ページングを通じて，活動中のバックグラウンド・ジョブおよびシステム機能と，主記憶装置を共有している．TSOでは，多重のユーザが単一のTSOリジョンに割当てられるであろう．**主タイム・スライス** (major time slice) の終りで，TSOのリジョンを占めていたあるユーザは，別のもう1つのユーザがそのリジョンにスケジュールされうるように，不活動化され補助記憶装置へ移動される(スワップされる)であろう．VS2は，スワッピングの代りに，ユーザの主記憶装置駐在ページをVS2

の外部ページ記憶装置へ転送するために**ブロック・ページング**技法を用いる．これは LSQA 内に含まれているユーザのページ・テーブルを含む．変更ページのみをスワップするブロック・ページングは，MVT と比較したとき，データ転送がかなり少なくてすむ．ユーザが再活動化されたとき――それらの次の主タイム・スライスに対してスケジュールされたとき――，それらの以前に駐在したページの集合が主記憶装置内にブロック・ページ化される．

VS2 の外部ページ記憶装置，すなわちページング・データ・セットは，多様な IBM の直接アクセス記憶装置上に駐在しうる．バッチ環境では，あまりにも活動リジョンが多すぎると高ページング率，そしておそらくスラッシングを引き起こすこともありうるが，普通，低ページング率が期待できる．外部ページ記憶装置のための装置の選択はそれほど重大ではない．しかし，TSO システムは結局のところページング率が高くなる．タイムシェアリング環境では，外部ページング記憶装置のための装置の選択は性能，特にささいな(trivial)トランザクションに対する応答時間に，重要な影響を及ぼしうる．

これまでの VS2 の議論はリリース 1，すなわち AS(1) システムに関連したものであって，これは，その先行システム，OS/MVT と良好な互換性を有している．この VS2 の最初の型以後，VS2 のアーキテクチャはリリース 2 型でかなり拡張された．VS2-Rel2 は AS(n) システムである．あるシステム要素および各 VS2 ユーザ――バッチあるいは TSO――は，それら自身の 16m バイトのアドレス空間を持つ．したがって，オペレータが新しいジョブ・イニシエータを開始するたびに，新しいアドレス空間が創成される．TSO ユーザがログ・オンするたびに，そのユーザは自分自身のアドレス空間を得る．また，ある VS2-Rel2 サブシステムは，それら自身の AS 内に駐在する．これは，HASP ジョブ・エントリ・サブシステム(JES2)および，Rel2 では JES3 と呼ばれる，**アタッチト・サポート・プロセッサ**(Attached Support Processor: ASP)にあてはまる．VS2-Rel2 では，リジョン要求は除去される．各 VS2 アドレス空間の構造は**図 11-6.4** に示すものと同じである．その核は主記憶装置に固定され，1 組の共通ページ・テーブルによって，各アドレス空間内に写像される．この技法は 11-3 節に述べられた．**共通エリア**は同じ様式で写像される．残りのエリア，すなわち**個人用**(Private)**エリア**は，そのアドレス空間に割当てられたシステム機能，バッチ・ジョブ，あるいは TSO ユーザによって制御される．これは典型的には約 8～10m バイトに相当する．理論的には，VS2-Rel2 システム内に共存しうるアドレス空間の数には制限がない．しかし，CPU の能力と主記憶装置のサイズによって実際的な制限が課せられる．

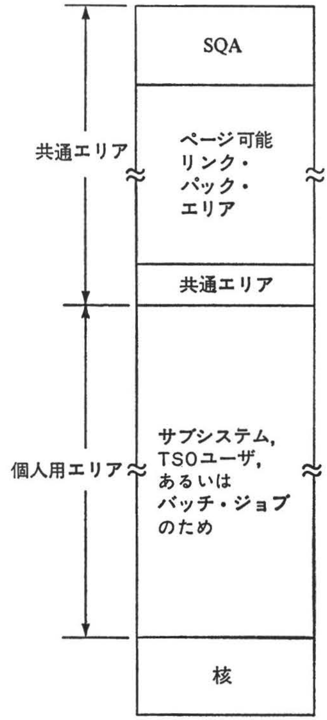


図 11-6.4 おのおののVS2-Rel2
アドレス空間の構造

VS2 システム，特に VS2-Rel2 は，OS/MVT との互換性を出発点とし，さらにそれからの有意義な脱却を示している．大きなアドレス空間と仮想記憶装置はシステム設計上，操作上，プログラミング上の多くの利点を与える．しかし次のようないくらかの犠牲もある．

1. 主としてページング・スーパバイザの付加のため，核のサイズが増加する．
2. アドレス翻訳は実行時間にかなりのオーバヘッドを付加するが，これは高速翻訳ハードウェアによって幾分縮少される．
3. チャネル・プログラム翻訳はソフトウェア (IOS) によって行なわなければならない．というのはシステム/370 のチャネルは翻訳ハードウェアを持っていないからである．
4. 外部ページ記憶装置と呼ばれる，ある補助記憶装置がデマンド・ページ

ングのために必要である．

仮想記憶装置はプログラマ，オペレータ，システム設計者，そしてユーザに対してより容易なシステム利用を許すという点で，仮想記憶装置の利点はこれらの犠牲を正当化する傾向がある．

11-7　IBM 仮想マシン(VM)システム [AO 1, AO 2, AU 1, BA 1, BA 2, CA, FI, GO 1, GO 2, GO 3, IB 7, MA 1, ME, PA 1, PA 2 参照]

VS2-Rel2 では，各ユーザは個人用アドレス空間を持ち，他の実行ユーザと CPU を共有し，そして割当てられた 1 組のリソースを制御する．ある意味で，各ユーザは個人用マシンを持つ．しかし，このマシンはシステム/370 ではない．というのはあるシステム/370 の命令は特権扱いにされており，VS2 制御プログラムにのみ利用可能であり，各ユーザには利用可能ではないからである．これは，ユーザが開発するプログラムとシステムの型にある制限を課す．**仮想マシン・ファシリティ**(Virtual Machine Facility)/370(VM/370)システムでは，各ユーザには，アドレス空間，1 組の I/O リソース，および CPU 時間の分前が与えられる．またユーザは**擬似特権状態**で実行しうる．その結果，仮想システム/370 は，現実のシステム/370 で実行可能ないかなるプログラムあるいはシステムも実行しうる．**図 11-7.1** は，単にユーザ・プログラムではない，複数のオペレーティング・システム全体をマルチプログラミングできる，VM 編成を示している．

VM/370 はシステム/360 モデル 67 のために開発されたコントロール・プログラム/67(CP/67)の後継システムである．VM/370 では，各端末ユーザは仮想記憶装置，仮想 I/O 装置，および CPU(VM/370 の制御下にあるものと同じ CPU モデル)から成る 1 つの仮想マシン構成を持つ．仮想 I/O 構成はテーブルを通じて実装置あるいは実装置の分割に写像される．たとえば，仮想カード・リーダあるいはプリンタは，1 つのディスク"域"によって表わしうる．仮想直接アクセス装置は，全ディスク装置あるいは，**ミニディスク**と呼ばれる概念の，1 ディスク域によって写像されうる．しかし，仮想テープ装置は実テープ装置によって 1 対 1 に表わさなければならない．実マシンと同様に，各仮想マシンは制御卓装置を必要とする．VM/370 では，御制卓はユーザの端末である．VM/370 に"ログ・オン"の後，ユーザは，もし適切な構成を有しているなら，実システム/370 で実行しうるいかなるシステムの IPL も行なうことができる．運転中，オペレータが自分の端末(VM システム制御卓)を通じ一組

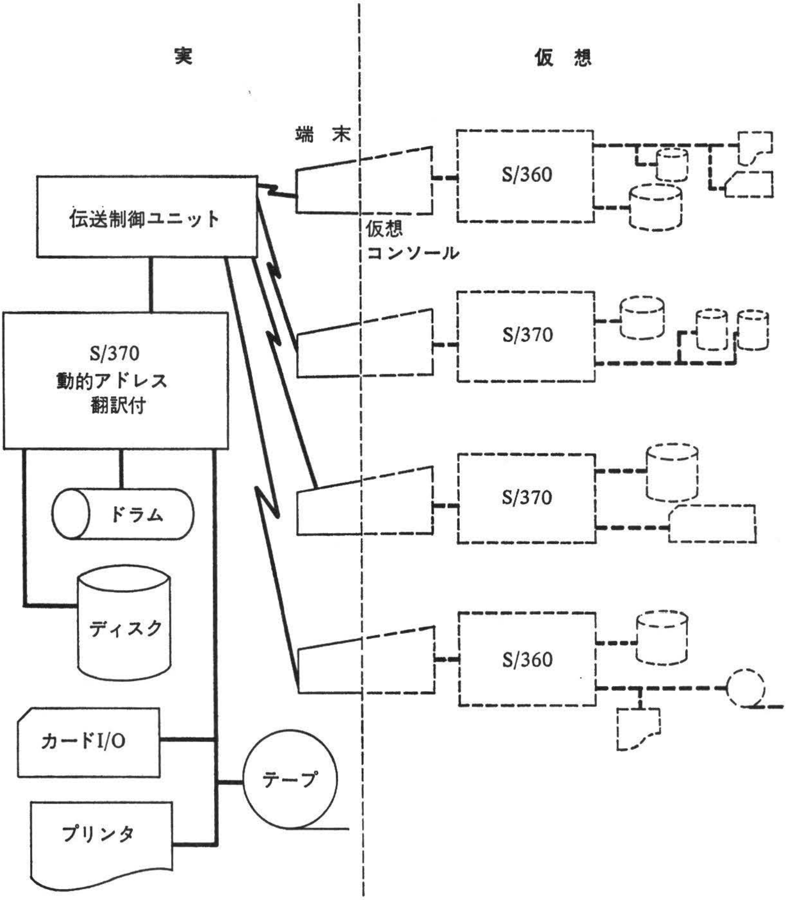


図 11-7.1 IBM VM/370 システムにおける仮想-実マシンの関係

の特別なコマンドによって通信するのと同様に，ユーザも通信しうる．

これらの機能はすべて VM/370 制御プログラムによって可能となるが，それはシステム運転中主記憶装置に固定される．制御プログラムは，仮想記憶装置と仮想 I/O 装置のためのすべての写像，仮想カード・リーダとプリンタのためのスプール・システムを制御し，CPU のスケジューリングを制御し，すべての特権命令および仮想割込を途中で捕えてそれにサービスする．この最後の機能によって仮想マシンは擬似特権状態で動作できる．特権命令が仮想マシン内で出されると，その命令を VM/370 へ渡すための割込が引起こされ，

VM/370 でサービスが行なわれた後，制御がユーザの仮想マシンに返される．同じ形式のサービスが仮想マシン割込に対して提供される．

VM/370 システムでは，数多くの VM が並行して走ることができる．たとえば，1つの仮想マシン内で OS/MVT，別のマシン内で VS2，そしてその他のマシン(複数)内で多くのタイムシェアリング・ユーザがそれぞれ1つの**会話型モニタ・システム**(Conversational Monitor System；略して **CMS**)の制御の下で走る．CMS は VM 環境で良好な性能を持つように設計された簡単だが効率のよいオペレーティング・システムである．VM/370 ユーザが CMS を IPL すると，そのユーザは多様なシステム/370 プログラムの機能を自分の思いのまま持てる．プログラムとテキストは端末から入力し，編集し，個人用ユーザ・データ・セットにストアすることができる．ANS COBOL, PL/I, および FORTRAN を含む，多くのシステム/370 のコンパイラは CMSの下で実行されうる．プログラムの実行テストおよびデバッグ機能もまた利用可能である．

VM/370 の機能は次のような多様な並行する活動を考慮に入れている．

1. オペレーティング・システムの開発と保守
2. 複雑なシステムのテスト
3. インタラクティブ・プログラムの開発

しかし，VM/370 には1つの重大な欠点――実行速度――がある．仮想マシン環境を提供することによりもたらされるオーバヘッドは，OS/MVT のような複雑なシステムの実行速度の低下という形で表われる．VS/370 下では，OS/MVT は実システム/370 上のたった2分の1の速度で実行することになってしまうかもしれない．これは，バッチ処理実行環境における VM/370 の有用性を制限する傾向がある．性能の濫用を少なくするのを助けるために，VM システムは多くの特徴を持っており，それらによってある装置では，特定の仮想マシンに主記憶装置と CPU の利用可能性を保証することによって，えこひいきする，すなわち，それを特権仮想マシンにすることができる．

11-8　MULTICS システム [BE, CO 1, CO 2, DA, OR, SA 参照]

Multics(Multiplexed Information and Computing Service) は MIT と General Electric Company によって共同で設計された．ハードウェアは基本的には GE645 で，それにシステム設計環境の仮想記憶装置にとって必要なハードウェアを付加している．オペレーティング・システムは複雑であり，その

プログラムを高水準言語(PL/I)で書くことを決定したのは注目すべきである.

Multics の主要目的は，ユーザがシステム内のすべての情報を(適切な安全性と保護の防護手段のもとで)共有することができるような，一般的なタイムシェアリングと仮想記憶装置の設備を築くことであった．この共有目的は Multics の最も興味ある特徴の多くにとって中心をなすものである．それは，設計者からの次の引用によって最も よく述べられている.

“多くの現代のシステムはある程度の共有を許している．普通は，2次記憶装置内に保持されたファイルにストアされている情報の入力と出力を経由して，多くのユーザがデータを共有することができるようにすることで，共有が達成されている．しかし，セグメンテーションの利用によって，Multics はすべての情報のユーザおよびシステム・プログラムによる直接的なハードウェア・アドレッシングを，その物理的な記憶装置のロケーションとは独立に，提供する．情報はセグメント内にストアされるが，各セグメントは潜在的に共有可能であり，またそれ自身の，独立な属性であるサイズとアクセス権を持っている.”

Multics は AS(n) 仮想記憶装置機構を用いる．各ユーザは最大 2^{18}(262, 144) セグメントまでを含みうるセグメント方式アドレス空間を持つ．セグメントのサイズは可変であり，2^{18} 語(およそ 1 m バイトに等しい)まで大きくなりうる．そのような大きなアドレス空間は，ユーザに対して多くのプログラムおよびデータ・セグメントを単一のアドレス空間に写像することを許すという目的に便宜を図るために，選択された．システムあるいはその他ユーザは多くのそのようなセグメントを共有しうる．実際にはユーザは比較的少ないセグメントで操作するかもしれないが，そのアドレス空間により多くのセグメントを付加する潜在能力はほとんど常に利用可能である．Multics がセグメンテーションのみを使っていたとすれば，システムは主記憶装置でのフラグメンテーションの問題を経験したであろうし，またセグメントが大きい場合には，主記憶装置内に共存するセグメントを比較的少ししか収容できなかったであろう．これらおよびその他の理由のために，Multics はセグメントを 1024 語のページに分割し，デマンド・ページング戦略を用いてページを割当てる.

GE 645 は Multics の要求のために特別に設計されたどちらかといえば複雑な写像ハードウェアを含む．原理的には，詳細を問わなければ，それは前に議論された写像機構に類似している．Multics の最も興味ある部分を短く要約するに際して，そのユニークなあるいはほぼユニークといえる特徴に議論を集中

しよう．

Multics の各ユーザのアドレス空間は**ディスクリプタ・セグメント**(descriptor segment；略して **DS**)によって写像される．DS の長さは 1 つのアドレス空間内に写像されるセグメントの総数に依存し，各セグメントに対して 1 つのエントリを持つ．あるユーザが活動中のとき，GE645 の**ディスクリプタ・ベース・レジスタ** (**DBR**) は，そのユーザの DS の主記憶装置上のロケーションを識別する．GE645 プロセッサにおけるすべてのアドレスは整数対 $[s, i]$ から成っている．ここに s はセグメント番号を識別し，i はそのセグメント内で参照されるロケーションを識別する．要素 i は，さらに 1024 を法とする演算を用いて，ページ番号 p と語ロケーションに変形される．GE645 内でのアドレス翻訳は，**図 11-8.1** に示す過程に従う．DS 自身もページ・テーブルによって写像されるので，ページ化されうることに注意しよう．これは重要なことである．というのはアドレス空間内のセグメントの総数は非常に多くなりうるので，結果的に DS が大きくなるからである．

Multics のスーパバイザは凝っていて複雑である．それはセグメント制御，

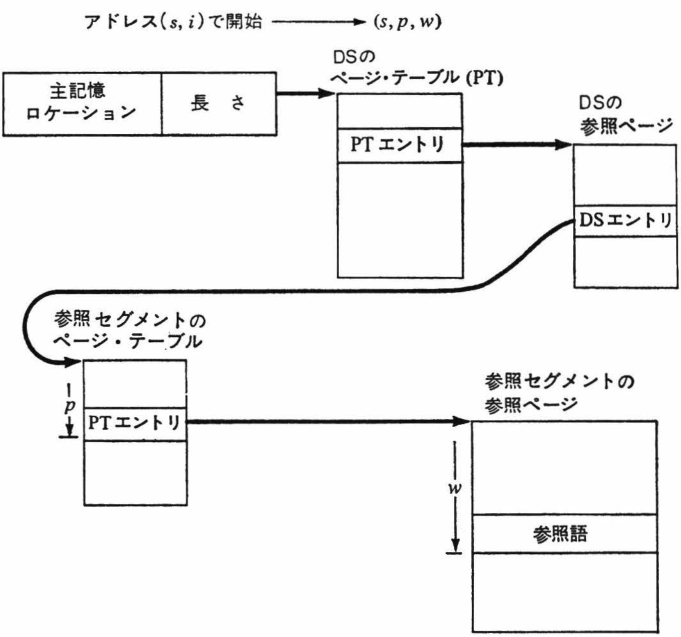


図 11-8.1　Multics のもとでの GE 645 におけるアドレス翻訳過程

ページ制御のための機能や，保護機構を用いてセグメントの割当てを制御するディレクトリ制御機能を含んでいる．スーパバイザはセグメント共有機構を用いて各ユーザのアドレス空間に写像されるが，その機構では各ユーザのスーパバイザ・セグメントのエントリは1組の共通のページ・テーブルを指している．実際には，セグメント共有のためのこの機構はシステムの至るところで使われている．スーパバイザは各アドレス空間に写像されているので，ユーザはスーパバイザ機能のサービスを要求することができるが，一方では，スーパバイザはリング保護機構によって，認められていない参照に対しては保護される．ページ制御機能を除いて，ほとんどのスーパバイザ機能はページ化されている．

他のページング・システムと同様，Multics もデマンド・ページング・アルゴリズムによって主記憶装置を制御する．ページは参照された際にまだ存在していないときのみ主記憶装置に持って来られる．置換えアルゴリズムが必要なときには，LRU アルゴリズムがページ選択のために呼ばれる．また Multics はページ・テーブルの空間を維持するために，セグメントを不活動化および再活動化する．主記憶装置の1つの固定化エリアが活動中のユーザ・セグメントを写像するページ・テーブルのために使われる．このエリアは今まで参照されてきたすべてのセグメント――それらの多くは現在**アイドル**かもしれない――を写像するのに必要な空間より小さい．テーブルのために不必要な空間を使うのを避けるために，ページ・テーブルは主記憶装置からページ・アウトまたはそれにページインされうる．新しく参照されたセグメントに対してページ・テーブルの空間が利用可能でないときは，無活動セグメントが，主記憶装置内に駐在するそれのページ数に基づいて，置換えのために選ばれる．希望的には，主記憶装置内にページを全く持たない無活動セグメントもいくつかあるであろう．それからシステムは主記憶装置からその選ばれたセグメントのページテーブルを移動し，そのセグメントを使用していたすべてのアドレス空間でこの不活動化を指示する．そして新たに参照されたセグメントを活動化する．実際，Multics はページ・テーブルの置換えに対して LRU アルゴリズムをインプリメントしようとしている．

Multics に対するこの入門的概論で，そのユニークな特徴，主としてプログラムとデータの共有を強調しようとしてきた．多くの詳細および他の仮想記憶装置システムに対する類似点を省略した．Multics はタイムシェアリングが優位を占める環境にある MIT で数年間成功裡に利用されてきている．

演　習　問　題

1. (a) AS(1)と AS(n) 仮想記憶装置システムの間の差異は何か.
   (b) AS(1)に比較して, AS(n) システムをインプリメントするためにどんな付加的なハードウェアあるいはソフトウェアが必要とされるか.
   (c) AS(1)の編成はどんな状況にもっとも適しているか.

2. 次のアーキテクチャの特徴の写像への効果(ハードウェアとソフトウェアおよびテーブルのための主記憶装置) を簡潔に論ぜよ. 固定されたサイズのアドレス空間を仮定する.
   (a) ページ・サイズ
   (b) セグメント・サイズ
   (c) セグメント・ページの編成とページのみの編成

3. 仮想記憶装置システムにおけるページング・スーパバイザの機能は何か.

4. 仮想記憶装置システムでは何故全域ページ配置戦略が用いられるのか.

5. 仮想記憶装置システムにおいてページを固定化することに対する理由をいくつかあげよ.

6. プリページング(あるいはコンテクスト・ページング)をデマンド・ページング技法と対比せよ. プリページングに固有な危険は何か.

7. 補助記憶装置の巨大 1 対 1 管理を外部ページ・テーブルの利用と対比せよ. 1 対 1 手法は AS(n) 環境に対して適当であるだろうか. それはなぜか.

8. AS(1) システムに対してどんなプログラム・ローディング技法が利用できるか. これは通常のマルチプログラミング環境でのプログラム・ローディングとどのように対比されるか.

9. 単一ユーザの観点から, AS(1) システムにまさる AS(n) の利点は何か.

10. AS(n)のインプリメンテーションにおいて, スーパバイザおよび他のシステム・プログラムはどのように共有されうるか. スーパバイザの物理的写しはいくつ存在するか.

11. AS(n)システムで創成されるアドレス空間の数を制限するのはどんなパラメタか.

12. IBM (およびほとんどの他のシステム) はI/O チャネルによる主記憶装置の参照を写像するのにソフトウェアのみを用いる. そのような参照をCPU の参照に対して行なわれるのと同様な方法で取扱うことの困難さを論ぜよ.

13. チャネル・プログラムのソフトウェア翻訳にとって必要な機能を述べよ. この過程の間どんなページが固定されねばならないか. それはなぜか.

14. チャネル・プログラムの翻訳に要するオーバヘッドを縮少するための2つの手法は何か.

15. (a) 仮想記憶装置の環境における**スラッシング**と呼ばれる状態について述べよ. どんな条件がスラッシングを引き起こさせるか.
 (b) スラッシング・モニタのいくつかの典型的な方策を簡潔に述べよ.

16. ブロック・ページングをデマンド・ページング技法と対比せよ. ブロック・ページングはどのようにして効果的に用いることができるか.

17. OS/VS2(Rel1)とOS/MVT の間の類似性と差異について述べよ.

18. OS/VS2 Rel1 とOS/VS2 Rel2 において記憶装置保護はどのようにして達成されているか.

19. TSO ユーザはVS2 Rel1 でどのように"スワップ"されているか.

20. システム・プログラムはVS2 Rel2 においてどのように共有されるか.

21. (a) 各VM/370 ユーザに提供される仮想マシンの基本的性質は何か.
 (b) (a)からどんな利点が生ずるか.
 (c) VM/370のアーキテクチャとユーザ・ファイルおよびプログラムの共有との両立性について論ぜよ.

22. Multics システムにおいて各ユーザに多数のセグメントを提供する目的は何か.

23. Multicsにおけるアドレス翻訳の間でのディスクリプタ・ベース・レジスタの目的は何か.

24. Multicsはページ・テーブルをページ化するためにどんな選択過程を用いるか.

25. この章に記述されていない仮想記憶装置システム，たとえばTSS/360, Burroughsなどに対して，その目的，機能，アルゴリズムなどについて述べた報告書を書け.

26. OS/MVTをOS/VS2と比較せよ．利用可能な機能，使用容易性，システムオーバヘッド(システム・プログラムのサイズおよびスーパバイザ状態での時間を含む)などを考慮すること.

参　考　文　献

1. [AD1] Adair, R. and Y. Bard: CP-67 Measurement Method, IBM Corp. Cambridge Scientific Center, *Report No.* G320-2072, May, 1971.
2. [RA2] Adair, R.J., R.U. Bayles, L.W. Comeau, and R.J. Creasy: A Virtual Machine System for the 360/40, IBM Corp. Cambridge Scientific Center, *Report No.* 320-2007, May, 1966.
3. [AN] Anacker, W. and C.P. Wang: Performance Evaluation of Computing Systems with Memory Hierarchies, *IEEE Transactions on Computers*, EC-16, No. 6, pp. 764-773, December, 1967.
4. [AU1] Auroux, A. and C. Hans: Le Concept de Machines Virtuelles, *Revue Francaise d'Informatique et de Recherche Operationelle* 15, no. B3, pp. 45-51, December, 1968.
5. [AU2] Auslander, M.A. and J.F. Jaffe: Functional Structure of IBM Virtual Storage Operating Systems Part I: Influences of Dynamic Address Translation on Operating System Technology, *IBM Systems Journal*, vol. 12, no. 4, pp. 368-381, 1973.
6. [BA1] Bairstow, J.N.: Many from One: The "Virtual Machine" Arrives, *Computer Decisions*, vol. 2, no. 1, pp. 29-31, January, 1970.
7. [BA2] Bard, Y., B.H. Margolin, T.I. Peterson, and M. Schatzoff: CP-67 Measurement and Analysis, I: Regression Studies, IBM Corp. Cambridge Scientific Center, *Report No. G320-2061, June, 1970.*
8. [BE] Benoussan, A., C.T. Clingen, and R.C. Daley: The Multics Virtual Memory–Concepts and Design, *Comm. ACM*, vol. 15, no. 5, May, 1972, p. 308.
9. [BR] Brawn, B. and F. Gustavson: Program Behavior in Paging Environment, *Proc. AFIPS 1968 FJCC*, vol. 33, pp. 1019-1032, 1968.
10. [BU] Buzen, J.: Optimizing the Degree of Multiprogramming in Demand Paging Systems, *IEEE Computer Society Conference*, Boston, Mass., pp. 141-142, September 22-24, 1971.
11. [CA] Callaway, P.: Performance Considerations for the Use of the Virtual Machine Capability, IBM Corp. Thomas J. Watson Research Center, Yorktown Heights, N.Y., *Report RC*-3360, May 12, 1971.
12. [CO1] Corbato, F.J.: A Paging Experiment With the MULTICS System, *Massachusetts Institute of Technology Project MAC Memorandum*, MAC-M-384, July 8, 1968.

13. [CO2] Corbato, F. and V.A. Vyssotsky: Introduction and Overview of the MULTICS System, *AFIPS Conference Proc., Fall Joint Computer Conference 27*, Part I, pp. 185-196, 1965.
14. [DA] Daley, R.C. and J.B. Dennis: Virtual Memory, Processes, and Sharing in MULTICS, *Comm. ACM*, vol. 11, no. 5, pp. 306-312, May, 1968.
15. [DE1] DeMeis, W.M. and N. Weizer: Measurement and Analysis of a Demand Paging Time-Sharing System, *Proc. of the 24th National Conference on Systems Science*, University of Hawaii, pp. 449-451, January 12-14, 1971.
16. [DE2] Denning, P.J.: Thrashing: Its Causes and Prevention, *AFIPS Conference Proc., Fall Joint Computer Conference*, vol. 33, pp. 915-922, 1968.
17. [FI] Field, M.S.: Multi-Access Systems–The Virtual Machine Approach, IBM Corp. Cambridge Scientific Center, *Report No.* 320-2033, September, 1968.
18. [FU] Fuchel, K. and S. Heller: Considerations in the Design of a Multiple Computer System with Extended Core Storage, *Comm. ACM*, 11, no. 5, pp. 334-340, May, 1968.
19. [GE] Gelenbe, E.: Optimum Choice of Page Sizes in a Virtual Memory with a Hardware Executive and a Rapid-Access Secondary Storage Medium, *ACM SIGOPS Workshop on System Performance Evaluation*, Harvard University, pp. 321-336, April 5-7, 1971.
20. [GL] Gluck, S.E.: Impact of Scratchpads in Design: Multi-functional Scratchpad Memories in the Burroughs B8500, *AFIPS Conference Proc., Fall Joint Computer Conference*, 27, Part I, pp. 661-666, 1965.
21. [GO1] Goldberg, R.P.: Hardware Requirements for Virtual Machine Systems, *Proc. of the 24th Hawaii Conference on Systems Science*, University of Hawaii, pp. 449-451, January 12-14, 1971.
22. [GO2] Goldberg, R.P.: Virtual Machine Systems, *Massachusetts Institute of Technology, Lincoln Laboratory Report*, MS-2687, September 4, 1969.
23. [GO3] Goldberg, R.P.: Virtual Machines: Semantics and Examples, *IEEE Computer Society Conference*, Boston, Mass., pp. 141-142, September 22-24, 1971.
24. [HA1] Hatfield, D.J.: Experiments on Page Size, Program Access Patterns, and Virtual Memory Performance, *IBM Journal of Research and Development*, vol. 16, no. 1, pp. 58-66, 1972.
25. [IB1] IBM Corp.: "IBM System/370 System Summary," No. GA22-7001, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
26. [IB2] IBM Corp.: "IBM System/370 Principles of Operation," No. GA22-7000, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
27. [IB3] IBM Corp.: "IBM System/370 Advanced Function Bibliography," No. GC20-1763, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
28. [IB4] IBM Corp.: "OS/VS1 Planning and Use Guide," No. GC24-5090, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
29. [IB5] IBM Corp.: "OS/VS2 Planning and Use Guide," No. GC28-0600, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
30. [IB6] IBM Corp.: "Introduction to OS/VS2 Release 2," No. GC28-0661, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
31. [IB7] IBM Corp.: "IBM Virtual Machine Facility/370–Introduction Manual," No. GC20-1800, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
32. [IB8] IBM Corp.: "IBM Virtual Machine Facility/370 Planning and System Generation Guide," No. GC20-1801, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.
33. [IB9] IBM Corp.: "IBM Virtual Machine Facility/370: Release 2 Planning Guide," No. GC20-1814, IBM Corp. Data Processing Division, White Plains, N.Y., 10604.
34. [IB10] IBM Corp.: "Introduction to Virtual Storage in System/370," No. GR20-4260, IBM Corp. Data Processing Division, White Plains, N.Y. 10604.

35. [JO] Johnson, O.W. and J.R. Martinson: "Virtual Memory in Time-Sharing System/360, TSS/360 Compendium," IBM Corp. Data Processing Division, White Plains, N.Y., 1969.
36. [LE] Lett, J.S. and W.L. Konigsford: TSS/360: A Timeshared Operating System, *AFIPS Conference Proc., Fall Joint Computer Conference*, 33, Part I, pp. 15-28, 1968.
37. [MA1] Madnick, S.E.: Time-Sharing Systems: Virtual Machine Concept vs. Conventional Approach, *Modern Data*, vol. 2, no. 3, pp. 34-36, March, 1969.
38. [MA2] Martinson, J.R.: Utilization of Virtual Memory in Time-Sharing System/360, IBM Corp. Systems Development Division, Yorktown Heights, N.Y., *Technical Report TR* 53.0001, October 28, 1968.
39. [ME] Meyer, R.A. and L.M. Seawright: A Virtual Machine Time Sharing System, *IBM Systems Journal*, vol. 9, no. 3, 1970.
40. [ORG] Organick, E.I.: "The Multics System," MIT Press, Cambridge, Mass., 1972.
41. [PA1] Parmelee, R.P.: Preferred Virtual Machines for CP-67, IBM Corp. Cambridge Scientific Center, *Report No.* G320-2068.
42. [PA2] Parmelee, R.P., T.I. Peterson, C.C. Tillman, and D.J. Hatfield: Virtual Storage and Virtual Machine Concepts, *IBM Systems Journal*, vol. 11, no. 2, 1972.
43. [SA] Saltzer, J.H.: A Simple Linear Model of Demand Paging Performance, *Comm. ACM*, vol. 17, no. 4, April, 1974.
44. [SM] Smith, J.L.: Multiprogramming Under a Page on Demand Strategy, *Comm. ACM*, vol. 10, no. 10, pp. 636-646, October, 1967.
45. [SC] Scherr, A.L.: Functional Structure of IBM Virtual Storage Operating Systems Part II: OS/VS2-2 Concepts and Philosophies, *IBM Systems Journal*, vol. 12, no. 4, pp. 382-400, 1973.

補遺 I

確率関数

いくつかの離散型確率関数

関数名	分布関数	平均値	分散	意味
二項	$B(k;\ n,\ p)$ $=C(n,k)\,p^k(1-p)^{n-k}$	np	$n(p1-p)$	n 回試行のうち k 回成功する確率；$p=1$ 回の試行で成功する確率
多項	$P(k_1,\cdots,k_n;p_1,\cdots,p_n)$ $=m!\prod_{i=1}^{n}\frac{p_i{}^{k_i}}{k_i!}$ ただし $\sum_{i}^{n}k_i=m$	mp_i	$mp_i(1-p_i)$	m 回の試行で1番目のタイプが k_1 回，2番目のタイプが k_2 回，・・・出される確率
幾可	$G(k;\ p)=p(1-p)^k$	$\frac{1-p}{p}$	$\frac{(1-p)}{p^2}$	$k+1$ 回目の試行で初めて成功する確率
ネガティブ二項 (Pascal)	$N(k;\ r,\ p)$ $=C(r+k-1,\ k)$ $\times p^r(1-p)^k$	$\frac{r(1-p)}{p}$	$\frac{r(1-p)}{p^2}$	$r+k$ 回目の試行で r 回目の成功が現われる（k 回失敗する）確率
Poisson	$P(k;\ \lambda)=\frac{\lambda^k e^{-\lambda}}{k!}$	λ	λ	k 回成功する確率

いくつかの連続型確率関数

関数名	確率密度	平均値	分散
指数 $0\leq x<\infty$	$p(x)=\lambda e^{-\lambda x}$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$
a から b までの一様 $a\leq x\leq b$	$p(x)=\frac{1}{b-a}$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
正規 $-\infty\leq x\leq\infty$	$p(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{-(x-\mu)^2/2\sigma^2}$	μ	σ^2
標準正規 $-\infty\leq z\leq\infty$	$\phi(z)=\frac{1}{\sqrt{2\pi}}e^{-z^2/2}$	0	1

補　遺 II

連続型確率関数の原理

基　礎

連続型確率変数 x は連続体上の値をとることができる．それの性質の多くは，離散型確率変数の性質にきわめてよく似ている（3章参照）．x の値が x と dx との間にある確率は $p(x)dx$ であり，確率性の公理はこれら微小量すべての積分で表わされる．

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{1}$$

単に $p(x)$ ではなく $p(x)dx$ が，離散型確率関数に対応することに注意しよう．したがって，x が a と b の間にある確率は次のようになる．

$$P(a \leq x \leq b) = \int_a^b p(x)dx \tag{2}$$

累積分布は同じような積分になるが，ただ，下限は x の考えうる最小値で上限は自由変数 z となる．

$$P(z) = P(x \leq z) = \int_{-\infty}^{z} p(x)dx \tag{3}$$

式(3)(と微積分学)から，**確率密度関数**(probability density function)と呼ばれる $p(x)$ は累積分布の微分に等しいことに注意しよう．

$$p(z) = \frac{dP}{dz} \tag{4}$$

$p(x)dx$ は x が x と $x+dx$ の間にある確率であるから，

$$xp(x)dx = \text{重みをつけた } x \text{ の値}$$

である．積分によってこれらの和をとれば，結果は x の**期待値**(expection)あ

るいは**平均値**(mean)となる．

$$\mu_x = E(x) = \int_{-\infty}^{\infty} xp(x)dx \tag{5}$$

分散(variance)は，平均値との差の2乗に確率の重みをつけ，それを積分したものである．

$$\sigma_x = \int_{-\infty}^{\infty} (x - \mu_x)^2 p(x)dx \tag{6}$$

上に述べた原理を使えば離散型確率変数の性質の多くは連続型のそれによく似ている．

正　規　分　布

正規分布(ガウス分布あるいはラプラス分布とも呼ばれる)は確率論で最も重要な分布である．この理由は次の3つである：(1) 自然界の多くの過程はこの分布にあてはまるように思われる．(2) 正規分布は簡単な数表で表わすことができ，もっと厄介な数表を必要とするような他の分布を近似するのに用いられる．(3) 中心極限定理によれば，たいていの任意の分布からとった標本値の和の分布は正規分布になりやすい．

現実の問題では変数は x であるが，規準変量 z に変換すると非常に便利である．

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

ここに μ は x の平均値，σ は標準偏差である．標準正規分布は次のようになる．

$$\phi(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2} \tag{2}$$

x の値が平均値より小さい(大きい)と，z は負(正)の値となることに注意しよう．式(1)を使って x を z に変換すると，x の曲線は $x = \mu$ が $z = 0$ となるように平行移動され，$x - \mu$ の値は標準偏差で割られて縮小される．**図II-1**は，$\phi(z)$ が $z = 0$ について左右対称な釣鐘形の曲線になることを示している．言い換えれば，z の正の値は，負の値と同じ重みが付けられている．したがって，z の平均値は0である．また，標準正規分布の分散は1であり，それゆえ標準偏差も1になることも簡単に示される．

$$\mu_z = 0 \qquad \sigma_z = 1 \tag{3}$$

累積正規分布は**誤差関数**とも呼ばれ，ϕ の曲線下の部分の面積であり，次のようになる．

$$\Phi(t) = \Phi(z \leq t) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{t} e^{-z^2/2}dz \tag{4}$$

$\Phi(t)$ の値（またはこれに関係の深い値）が表に与えられている．表から $\Phi(t)$ の値を得る方法はいくつかある．たとえば，付録の表 C3 は 0 から t までの $\phi(t)$ 下の面積を与える（表の添図参照）．表の値を $T(t)$ と呼ぶことにすれば，累積正規分布との間には次の関係がある．

$$\begin{aligned}\Phi(t) &= 0.5 + T(t) \qquad t \geq 0 \text{ に対して}\\ &= 0.5 - T(t) \qquad t < 0\end{aligned} \tag{5}$$

また，$\Phi(a \leq t \leq b)$ は a から b までの $\phi(t)$ の面積なので次式が成り立つ．

$$\Phi(a \leq t \leq b) = \Phi(b) - \Phi(a) \tag{6}$$

式(1)から簡単な計算により，**図 II-1** での z 軸上の 1 の長さは x の正規曲線では標準偏差(σ)の長さに相当することがわかる．図からわかるように，中心を離れると面積は急激に減少し，たとえば面積の 95.45 パーセントは x の平均値の両側 2σ の範囲内にある．

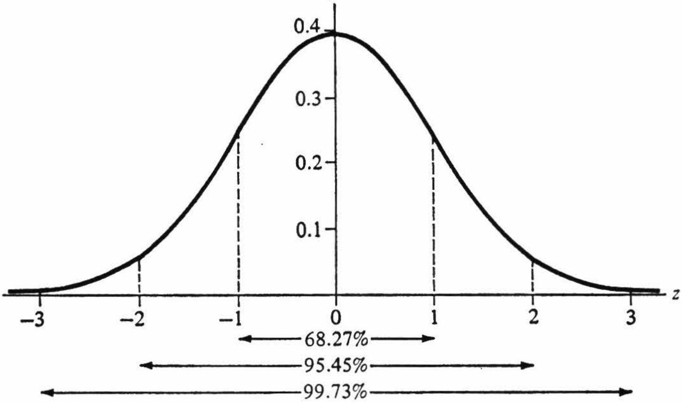


図 II-1　標準正規曲線，標準偏差の区間における面積の割合を示してある．

k と n が非常に大きく $(k-\mu)^3/\sigma$ が小さい場合，二項分布は正規分布によってかなり近似できる．主に興味があるのは，二項**累積分布**の標準正規近似である．近似式は次のようになる．

$$\sum_{j=k_1}^{k_2} C(n,\ j)p^j(1-p)^{n-j} = \Phi\left(y_2 + \frac{1}{2\sigma}\right) - \Phi\left(y_1 - \frac{1}{2\sigma}\right) \tag{7}$$

ただし

$$y_1 = \frac{k_1 - np}{\sigma},\quad y_2 = \frac{k_2 - np}{\sigma},\quad \sigma = \sqrt{np(1-p)}$$

である．

例：　成功する確率がそれぞれ $p=1/5$ である 100 回の Bernoulli 試行を考える．24 回以上成功する確率はいくらか？

解：

$$np = (1/5)(100) = 20;\ \sigma = \sqrt{20(0.8)} = 4$$
$$k_1 = 24;\ y_1 = (24-20)/4 = 1$$
$$k_2 = 100;\ y_2 = (100-20)/4 = 20$$

式(7)より右辺は

$$P(k \geq 24) = \Phi(20 + 1/8) - \Phi(1 - 1/8)$$

付録の表の値 $T(t)$ を使い，必要なら内挿を行なうと

$$\Phi(20 + 1/8) = 0.5 + T(20.125) = 0.5 + 0.5 = 1$$
$$\Phi(1 - 1/8) = 0.5 + T(0.875) = 0.5 + 0.3092 = 0.8092$$

ゆえに

$$P(k \geq 24) = 1 - 0.8092 = 0.1908$$

正規分布が非常に重要である理由の 1 つは次の定理であり，証明なしに掲げる(このような定理がいくつかある)．

中心極限定理

$X_1, X_2, \cdots, X_n$ が平均 μ，標準偏差 σ の同一分布に従う互いに独立な確率変数であるなら，和 $X_1 + X_2 + \cdots + X_n$ の累積分布は n が無限に増加するにしたがって標準正規分布に近づく．

$$\lim_{n\to\infty} P\left(\frac{X_1+X_2+\cdots+X_n-n\mu}{\sigma\sqrt{n}}\leq a\right)=\Phi(a)$$

この定理の注目すべき点は，それが確率変数の分布の形には無関係なことである．このことは非常に重要である，というのはわれわれは通常多くの“ミクロな”根元事象の組合された結果を観測するからである．これらの事象が互いに独立である時に組合せて和をとれば，中心極限定理によると観測された和は根元事象には無関係に正規分布に従う．たとえば，おのおのが毎日多数のジョブを依託している多数の計算機ユーザの集まりを考えよう．ユーザ1人・1月当りの総処理時間はおのおののユーザの処理時間の和である．個々のジョブの時間分布は正規分布でなくても，ユーザ1人当りの毎月の処理時間の分布関数は正規分布に従うであろう．

χ^2-検　定

応用範囲の広いこのノンパラメトリック検定は，任意の2つの度数分布の間の差がランダムな変動に帰因する確率を測る．まずこの理論を概観し，次いでχ^2-検定を行なうための計算手続きを与えよう．

それぞれの確率が $p_1, \cdots, p_n$ である n 点のアルファベットを持つ標本空間を考えよう．ある試行が m 回独立に行なわれ，その結果を標本点のラベルの長い系列として記録するとする．また，各点の出現回数を度数ベクトル f として記録する．そのようなベクトルの確率は多項分布で与えられる [3-15 節，式(7)]．

$$P(f_1, f_2, \cdots, f_n)=\frac{m!}{f_1!\,f_2!\cdots f_n!}p_1{}^{f_1}p_2{}^{f_2}\cdots p_n{}^{f_n} \tag{1}$$

ただし

$$\sum_{i=1}^{n} f_i = m \tag{2}$$

さて，実験的に得られた f_i の値を，いわば理論的なモデルと比較するものとしよう．このモデルの度数は，p_i を使って次のように見積られる．

$$f_i' = mp_i \tag{3}$$

通常これらは f_i と異なるので，どれぐらい異なっているかを調べるために，χ^2(カイ二乗)と呼ばれる偏差量を用いる．

$$\chi^2 = \sum_{i=1}^{n} \frac{(f_i - f_i')^2}{f_i'} \tag{4}$$

差 $f_i - f_i'$，したがって χ^2 は次の2種類の原因から生じる：(1) 仮定された理論的分布が観測された過程のよいモデルではないことによる真の偏差，(2) 任意の特定な度数ベクトル f はベクトル mp に等しい必要はないという事実による，ランダムな偏差．これらの分離を行なうために，われわれは**帰無仮説** H_0 という統計学者の考えを用いる．観測された度数ベクトル f は次のような意味において，理論的な度数ベクトル $f' = mp$ に一致する．

$$P_n(> \chi^2) < Q$$

これは，偏差量が χ^2 より大きい確率はある"信頼水準または有意水準" Q より小さいという意味である．ある与えられた水準 Q に対して不等式が満足されれば，われわれは帰無仮説を採択する．そうでなければそれを棄却する．

鍵となる解析問題は関数 $P_n(> \chi^2)$ の形である．これは非常に退屈な代数計算問題なので，ここでは理論を概説するにとどめる．式(1)の多項分布はまず多変量の正規関数で近似される．

$$P(f_1, f_2, \cdots, f_n) = \frac{1}{\sqrt{2\pi n}^{\,n-1}\sqrt{p_1, p_2, \cdots, p_n}} e^{-\chi^2/2} \tag{5}$$

この近似は，$f_i \gg 1$ の場合に正しい．与えられたデータに対して計算された χ^2 よりも大きな χ^2 を持つようなすべての f_i ベクトルの集合について P を積分する．この和，すなわち $P(\chi^2)$ の累積分布が，いろいろな χ^2 の値といろいろな**自由度** ν に対して付録の表C4に掲げられている．ν の値は，標本点の数から制約条件，すなわちデータから理論的分布のいろいろなパラメータを計算するのに使われる関係の数を減じたものである．例えば，$\sum p_i = 1$ であるから，この制約条件は自由度が高々 $n-1$ であることを意味している．データが理論的分布の平均値を近似するのに使われるならば，χ^2-表の $\nu = n-2$ の場合を用いる．

χ^2-検定が有効であるためには，以下の条件が満足されねばならない．(1) 各観測値は唯一つの度数区間に入らなければならない．(2) 観測された m 回の試行の結果は独立でなければならない．(3) 試行回数は大きくなければならない．初めの2つの条件は，χ^2-検定の基礎となっている多項分布のために必要である．3番目の条件は χ^2 の分布を通常の関数で近似するのに必要である．

さて，"一致の度合"を調べるために χ^2-検定を実際に行なう**計算手順**を要約しておこう．

前提： 次のような2つのデータ・ベクトルが与えられている．

（a） 測定された度数に一致すると考えられるある分布関数から得られる理論的な確率

（b） 測定された変数

測定された度数は理論的分布のある必要なパラメータ(平均値など)を計算するのに使われることに注意せよ．

設問： 2つのデータ・ベクトルの間の偏差はサンプリングにより生じた純粋にランダムな変動によるものかどうか．この問には，有意水準 Q で**そうである**か**そうではない**と答えなければならない．

手順：

1. （a） 帰無仮説 H_0 をたてる．これは次のような言明である：測定された分布は有意水準 Q で理論的分布に一致する；偏差はサンプリングに伴うランダムな変動に帰因する．

   （b） H_0 の対立仮説 H_1：帰無仮説を棄却する；偏差が大きすぎるので，実験的分布は有意水準 Q で理論的分布に一致しない．

2. f と p のベクトルから χ^2 を計算する．

$$\chi^2 = \sum_{i=1}^{n} \frac{(f_i - mp_i)^2}{f_i}$$

3. 自由度 df を決める．高々，$df = n - 1$ であるが，理論的分布の1つのパラメータを見積るのにデータが使われたならば，$df = n - 2$ となる；平均値と標準偏差のような2つのパラメータを見積るのに使われたならば $df = n - 3$ となる．

4. χ^2-表を見て，行の値 $= df$，列の値 $= Q$ の交点の欄の値を読む．これを x^2 と呼ぶ．これは，偏差が x^2 より大きいのはサンプリングだけが原因だと考えられる確率は Q であるという意味である．

5. 場合 1．$\chi^2 > x^2$ ならば帰無仮説を棄却する．

   場合 2．$\chi^2 \leq x^2$ ならば帰無仮説を採択する．

   1.の場合には，ステップ2.で計算された実際の偏差量が，ランダム・サンプリングの変動だけによると考えることができる量より大きい；2.の場合には，それは等しいかより小さい．たとえば5%の有意水準($Q = 0.05$)に対し1の場合があてはまると，測定された分布が理論的モデルに一致する可能性は20分の1以下である．これが一般的な基準である．帰無仮説が棄却される場合には，仮定されたモデルからの偏差の原因として

考え得るものを見つけるために，当然われわれは χ^2 に最も大きな影響を与えたデータの事例研究を行なうであろう．

χ^2-検定のような統計的検定を行なうための仮説を立てる際には，偽であると証明するのが最もたやすい仮説を帰無仮説として採用する．検定の過程で起こる可能性のある誤りは2種類あり，統計学では次のように呼ばれている．

タイプ 1. 帰無仮説が実際には真であるのに棄却される．
タイプ 2. 帰無仮説が実際には偽であるのに採択される．

実際的な意味では，有意水準 Q の値によってどのタイプの誤りが起きるか(または，仮説の採択において誤りが起きないか)が決まる．有意水準が厳しすぎると，タイプ2. よりタイプ1. の誤りの方が起きやすく，またその逆の関係も成りたつ．

索　　引

あ　行

い

う

え

お

す

せ

そ

た　行

ち

は　行

ひ

ふ

へ

ほ

ま　行

み

む

め

も

や　行

ゆ

《訳者略歴》

大野　豊
おおの　ゆたか

1946年　東京大学工学部機械工学科卒業
1946年　鉄道技術研究所技官
1962年　工学博士
1972年　京都大学工学部教授
現　在　京都大学工学部情報工学科教授
同情報処理教育センター長・工学博士

田畑孝一
たばたこういち

1963年　京都大学工学部電気工学科卒業
1968年　京都大学工学部助手
1973年　京都大学工学部助教授
現　在　京都大学情報処理教育センター助教授・工学博士

阿草清滋
あぐさきよし

1970年　京都大学工学部電気工学第二学科卒業
現　在　京都大学工学部助手・工学修士

久保正敏
くぼまさとし

1972年　京都大学工学部電気工学科卒業
現　在　京都大学工学部助手・工学修士

オペレーティング・システムの性能評価

昭和55年10月25日　初版発行

© 1980　日本コンピュータ協会

訳　者　大　野　豊
発行者　村　越　勝　弘
発行所　**日本コンピュータ協会**
東京都文京区本郷3丁目5-5
〒113　電話(03)813-6929番
発売元　**科学技術出版社**
東京都文京区本郷3丁目5-5
〒113　電話(03)813-6920番

検印廃止

乱丁本・落丁本はお買求め書店にてお取りかえいたします.　Printed in Japan